applied sciences

MDPI

Article

A Configurable Rule-Based Platform for Attack-Triggered PCAP Segmentation and Session-Level Multi-Protocol Feature Extraction

Eru Choi, SeokMoo Heo, Dinh Tuan Anh, Jiwon Heo, Tran Quoc Dang and Min Choi *

Department of Information and Communication Engineering, Chungbuk National University, Cheongju 28644, Republic of Korea; realeru@chungbuk.ac.kr (E.C.); smoo.heo@chungbuk.ac.kr (S.H.); adinh26101@chungbuk.ac.kr (D.T.A.); j624041@chungbuk.ac.kr (J.H.); collide0412@chungbuk.ac.kr (T.Q.D.)
* Correspondence: mchoi@chungbuk.ac.kr; Tel.: +82-43-261-2480

Abstract

Post-attack traffic analysis usually chains separate tools for detection, PCAP splitting and feature extraction. This paper presents a configurable rule-based platform that integrates these steps. Stage 2 locates attack intervals with one of five detection modes and writes each interval as a self-contained PCAP file by byte offset; Stage 3 reconstructs bidirectional sessions and extracts 49 features, including TLS, DNS and HTTP fields and payload entropy. Only the TCP-flag mode (Mode 4) was evaluated, on CICDDoS2019 and two CICIDS-2017 captures. Session precision was 99.52% for SYN flood traffic. For PortScan and DoS, the cross-session rules classified 20,821 and 167,305 sessions without false positives, but precision over all Stage 3 sessions was 21.05% and 75.88%, and session recall was 13.1% and 66.2%. The DoS settings detected no attack in CSE-CIC-IDS2018 without retuning. A single-pass implementation of the segmentation produced the same attack-interval files as the original remainder-file loop and processed the 8.07 and 12.28 GiB captures end to end in 363 and 369 s with a peak resident memory of about 1.1 GB. The work is a system contribution for building forensic attack-interval datasets; its rules require tuning for each attack type and corpus.

Keywords: traffic analysis; network forensics; PCAP segmentation; feature extraction; dataset generation; multi-protocol analysis; attack detection; machine learning

1. Introduction

1.1. Research Background and Motivation

Owing to rapidly growing network traffic and increasingly sophisticated cyberattacks, network security analysis is more important than ever. In modern network environments, various types of anomalous behavior occur simultaneously, including distributed denial-of-service (DDoS) attacks, port scans, low-rate DoS attacks (Slowloris, Hulk, and others), brute-force attacks, and botnet communications. Each produces a different damage profile, such as loss of system availability, information leakage, or infrastructure takeover.

This upward trend in attack activity has been quantitatively confirmed by several security reports. DDoS attacks are a representative case. According to Cloudflare's DDoS threat report for the fourth quarter of 2025, as shown in Figure 1, 47.1 million DDoS attacks were mitigated on Cloudflare's own network in 2025, a 121% increase over 2024, and the largest attack observed by Cloudflare reached 31.4 Tbps [1]. This increase, however, is not limited to DDoS. Over the same period, attacks with different underlying mechanisms,

Check for updates

Academic Editors: Petros Nicopolitidis and Juan-Carlos Cano

Received: 1 September 2026
Revised: 18 October 2026
Accepted: 28 October 2026
Published: 3 November 2026

Copyright: © 2026 by the authors. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license.

such as port scans, low-rate DoS, and botnet-based C2 communication, have also increased. Because each attack exhibits a distinct traffic profile and detection cues, a single detection technique cannot cover all of them.

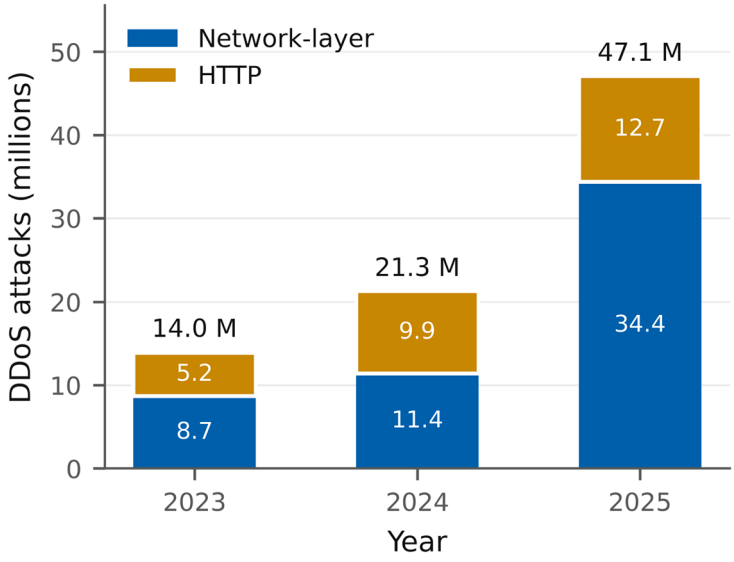


Figure 1. Trends in DDoS attacks by year and type (2023–2025).

The domestic situation shows the same pattern. According to the cyber threat trend report for the first half of 2025 published by the Korea Internet and Security Agency (KISA), and as shown in Figure 2, attempted DDoS attacks increased by 55.5% compared with the same period of the previous year [2]. Over the same period, server hacking (51.4%) and malicious code (11.1%) accounted for a large share of all reported security incidents, indicating that heterogeneous incident types coexist. In other words, modern network threats are unfolding not as an increase in one attack type but as the coexistence of diverse attacks, which suggests that detection tools specialized for a single attack type cannot respond effectively.

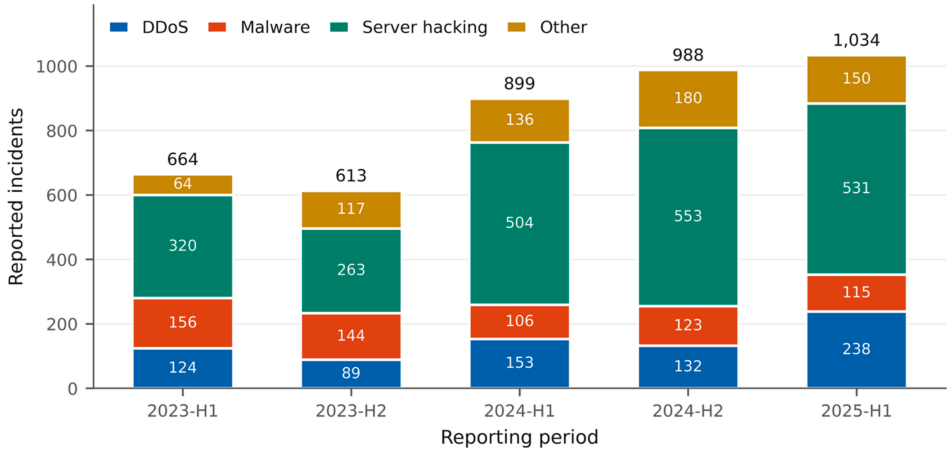


Figure 2. Yearly statistics of reported security incidents in Korea (Source: KISA).

Post-attack forensic analysis is essential for responding to such threats [3]. Forensic analysis plays a central role in understanding attack patterns, identifying vulnerabilities, and developing effective countermeasures. Such analysis is generally performed on captured packet traces (PCAP files), enabling precise post hoc investigation without affecting live network operations.

Wireshark, SplitCap, tcpdump, and Zeek are widely used for packet inspection, PCAP splitting, command-line capture and filtering, and traffic analysis, respectively [4–7]. Wireshark provides interactive protocol inspection [6]; however, the workflow evaluated in this study requires additional scripting or external tooling for automated attack-interval segmentation and feature-dataset generation. SplitCap splits PCAP files by network session, yet it offers neither attack-pattern-driven intelligent segmentation nor traffic feature extraction. tcpdump is a lightweight command-line tool that can be automated through scripting, but detecting complex attack patterns or extracting multidimensional features requires additional development. Zeek supports both offline PCAP analysis and live traffic monitoring and produces structured protocol logs [7,8]; however, it does not natively emit physically segmented attack-interval PCAP files or the specific 49-feature CSV schema used in this study.

Each of these tools excels at a specific function, yet no tool provides a coherent pipeline that covers detection, segmentation, feature extraction, and dataset generation. Network analysts must therefore use different tools in sequence depending on the type of attack, which increases analysis time, raises the likelihood of human error, and degrades reproducibility.

Conventional PCAP-based traffic analysis requires a combination of multiple tools and manual scripting. Typically, an analyst uses a separate filtering tool to identify intervals of interest in a large original PCAP: SplitCap or tcpdump for segmentation, CICFlowMeter for flow feature extraction, and Wireshark for in-depth protocol analysis. In this process, the analyst must write conversion scripts to reconcile mismatched output formats between tools and must connect intermediate artifacts manually at every step, which increases the repetitive workload and operational complexity of the toolchain.

The proposed framework integrates this multi-stage workflow into a single system. By uploading an original PCAP file, an analyst can automatically perform the entire process, from attack-interval segmentation and session reconstruction to the extraction of 49 multi-protocol features and CSV dataset generation. This eliminates the need to write scripts to connect tools or manually handle intermediate artifacts, and the resulting session features can be used as input to classifiers such as XGBoost [9] (Section 5.6.4). Interpreting the extracted features and fine-tuning the detection parameters still require the analyst's domain expertise. The contribution of this framework therefore does not lie in replacing that expertise but in automating the repetitive and tool-dependent preprocessing that precedes analysis. Beyond simply proposing new detection rules, this work consolidates and automates the workflow for large-scale traffic preprocessing and thereby substantially reduces the analyst's workload.

1.2. Research Objectives and Contributions

1.2.1. Research Objectives

The objective of this study is to develop an integrated, configurable rule-based framework for analyzing PCAP-based network traffic, to automate the entire process of attack detection, traffic segmentation, feature extraction, and dataset generation, and thereby to improve the efficiency of network security analysts. Specifically, the following goals are pursued:

- ✓ Development of a multi-mode detection framework capable of detecting diverse attack types (DDoS, PortScan, DoS, and others);
- ✓ Implementation of an algorithm that segments PCAP files with byte-offset-level precision;
- ✓ Construction of a comprehensive feature extraction framework based on per-protocol deep parsing;
- ✓ Full automation of the process with fast monitoring;

✓ Functional validation on multiple public datasets and quantification of the processing cost profile.

1.2.2. Research Contributions

This study makes the following contributions.

First, a configurable rule-based multi-mode detection framework is proposed. Unlike prior studies that developed detection systems specialized for a single attack type, the proposed system provides five detection modes covering ICMP flooding, ICMP payload patterns, port-based attacks, TCP flag anomalies, and hierarchical composite attacks. Using the same detection modes, a user can respond to different attack types such as DDoS, PortScan, and DoS simply by adjusting parameters (thresholds, flag combinations, target ports, and so on), without any separate reimplementation. Furthermore, the hierarchical detection mode can capture composite attack scenarios in which ICMP flooding and TCP flag anomalies occur simultaneously within a single analysis pass. Only Mode 4 is evaluated in this paper; Modes 1, 2, 3 and 5 are implemented but not evaluated.

Second, a byte-offset-based precise segmentation technique is implemented. Unlike conventional packet-level segmentation, the proposed system parses packets through jNetPcap callbacks [10] while maintaining the corresponding in-file byte offsets in application-level state, and the selected byte range is then transferred using the FileChannel.transferTo() method of Java NIO, which can avoid an explicit read/write loop but may perform a partial transfer that the caller must handle [11]; the implementation therefore repeats the call until the entire range has been written and then checks the size of each output file. During this process, the 24-byte PCAP global header is copied into each segmented file so that file integrity is guaranteed. The segmentation module uses a constant amount of memory regardless of the input file size. Owing to this property, large-scale datasets supplied in chunks can be processed sequentially within the heap ceiling, and this paper demonstrates the property on a 179 GiB workload. For a single large file, the session state of the detection stage grows with the number of unique flows in one continuous scan range (Equation (3)), and the remainder-file loop of the first implementation rescans the rest of the file after each attack interval (Equation (4)). A single-pass variant of the segmentation loop (Section 3.3) removes this rescan and produces the same attack-interval files; Section 5.5 reports both.

Third, a framework that extracts 49 network behavior features through per-protocol deep parsing is provided. Unlike CICFlowMeter, which offers features centered on TCP and UDP flow statistics, the proposed system extracts TLS, DNS, and HTTP application-layer features together with payload entropy within the same framework. In particular, the randomness of the payload is quantified by the Shannon entropy of Equation (11) and is used as an indicator for identifying encrypted or encoded traffic.

Fourth, a fully automated pipeline with fast monitoring is implemented. The entire process from file upload to feature extraction was automated through background tasks based on the @Scheduled annotation of the Spring Framework. Progress updates are transmitted using SSE, and the web-based interface provides interactive visualization, user-defined highlight conditions, and dynamic alerts. As a result, an analyst can intuitively carry out the whole process, from upload to result inspection, on a single screen without having to learn command-line tools, reflecting the focus on practical usability in the implemented system.

Fifth, the processing cost of the pipeline is measured on one machine and one storage device. For memory, the minimum heap that allowed a run to complete was measured by lowering the heap ceiling step by step; all three datasets completed with a 2 GB heap ceiling (CICDDoS2019 also with 1 GB), and at a 4 GB ceiling peak heap usage stayed between 0.85 and 1.23 GB. For time and disk, the remainder-file loop wrote 746 and 477 times the input size for the two single CICIDS-2017 captures, whereas the single-pass variant wrote only

the attack-interval files and reduced the end-to-end time from 5628 and 5603 s to 363 and 369 s with identical output. In a full-file benchmark on the same hardware (Section 5.6.2), Stage 3 alone took 0.68 to 0.78 times the processing time of CICFlowMeter v4.0 and used about seven times as much peak memory, or about four times with the optional idle timeout of Section 4.2.1; Zeek was the fastest of the three tools.

Beyond integrating these functions, the technical elements of this work are (1) iterative attack-triggered segmentation that emits self-contained PCAP files by byte offset with the global header preserved, together with the memory bound of Equation (3), under which end-to-end memory depends on the unique flows in one continuous scan range and not on the total input size; (2) a two-layer detection design in which Stage 2 triggers on patterns inside a single session and a separate cross-session step identifies attacks, such as PortScan, whose individual sessions resemble normal traffic; and (3) a hierarchical mode that records lower-layer TCP attack intervals inside an ICMP interval in a single pass. The platform is rule-based: the analyst selects a mode and sets its parameters for the attack under study, and the evaluation in Section 5 covers three attack types on two public datasets.

2. Related Work

This section reviews prior work related to the proposed system in three areas: PCAP analysis tools, network feature extraction techniques, and network anomaly detection systems.

2.1. PCAP Analysis and Segmentation Tools

Wireshark is the most widely used open-source packet analyzer and supports GUI-based live capture and deep protocol decoding. Because it focuses primarily on interactive packet analysis, however, it is not directly optimized for automated batch session segmentation of large PCAP files or for attack-interval detection. Because users must apply filters and extract data manually, it is inefficient for repeated analysis of large-scale datasets.

SplitCap is a Windows-only PCAP splitting tool developed by Netresec that supports session segmentation based on the 5-tuple (source and destination IP, ports, and protocol). It is fast, but it does not directly support precise byte-level segmentation of attack intervals (byte-offset splitting), and because it has no feature extraction capability for the segmentation results, a separate post-processing tool is required.

Zeek (formerly Bro) [7] is a network security monitoring framework that supports both offline PCAP analysis and live traffic monitoring [8] and automatically generates structured per-protocol logs such as connection logs (conn.log), DNS logs (dns.log), and HTTP logs (http.log). It can be extended through a scripting language, but extracting detailed TLS handshake fields (cipher suite lists, extension lists, SAN counts, certificate validity periods, and so on) in an integrated, per-session form requires additional user-defined scripts. In addition, binary-level segmentation of PCAP files is not provided by default.

tcpdump is a command-line packet capture tool that supports filtering through BPF (Berkeley Packet Filter) expressions. It is lightweight and fast, but it does not include high-level analysis capabilities such as session reconstruction, advanced statistical feature extraction, or attack detection.

Beyond these tools, a body of research has addressed the storage, indexing, and retrieval of large packet traces. The Time Machine records traffic selectively and supports retrospective retrieval around security events [12], whereas pcapIndex builds an external index over unmodified trace files to accelerate interval queries [13]. Payload attribution methods [14] and dedicated forensic storage infrastructures [15] pursue the same goal of locating relevant traffic within large archives, and recent work has examined scalable near real-time capture platforms [16] and software deep packet inspection for traffic analysis and anomaly detection [17]. These approaches, however, either keep the original trace intact

and depend on an auxiliary index, or target live capture. None of them emits self-contained, physically segmented attack-interval files that can be passed directly to a downstream feature extraction stage.

Each of these tools excels in its own domain, but none of them provides, as a single tool, a coherent pipeline that spans attack detection, traffic segmentation, feature extraction, and dataset generation.

2.2. Network Traffic Feature Extraction

CICFlowMeter is a Java-based network traffic feature extraction tool developed by the Canadian Institute for Cybersecurity. It provides more than 80 statistical flow features [18,19] and has been used to generate representative intrusion detection benchmark datasets such as CICIDS-2017 [20] and CSE-CIC-IDS2018. It concentrates, however, on statistical features at the TCP and UDP flow level and does not extract, by default, features spanning multiple protocol layers such as TLS handshake parsing, DNS query character distribution analysis, or per-packet Shannon entropy. In particular, the absence of feature extraction specialized for the ICMP protocol limits its usefulness for the precise analysis of ICMP-based attacks.

nDPI is an open-source deep packet inspection (DPI) toolkit developed by ntop that supports a broad and evolving set of Layer-7 protocols and extracts protocol metadata [21]. As a library, it requires application-specific integration to produce the fixed per-session CSV schema used in this study.

Joy is a network forensic tool developed by Cisco [22] that generates flow records in JSON form and supports TLS fingerprinting. It does not, however, include analysis specialized for ICMP traffic, hierarchical attack detection logic, or PCAP segmentation, and its repository has been archived since August 2026 and is no longer actively maintained.

The proposed system extracts 49 session features within a single pipeline, combining characteristics from multiple protocol layers (TCP and UDP flow statistics, TLS handshakes, DNS queries, HTTP metadata, and payload entropy and character distribution) that existing tools support only individually.

2.3. Network Anomaly Detection Techniques

In network anomaly detection research, statistics-based techniques such as entropy-based anomaly detection [23], rate limiting, and threshold-based rules have been widely used. They are relatively simple to implement and suitable for fast processing, but their adaptability to new attack patterns is limited.

Port-scan detection commonly exploits cross-flow patterns such as the number of distinct destination ports, flow size, TCP flags, and temporal aggregation [24,25]. Low-rate DoS attacks (Slowloris, Hulk, GoldenEye, and others) maintain connections while transmitting slowly, which makes volume-only detection difficult. Duration- and rate-related session features have therefore been used for detection [26,27].

The main limitations of prior work are as follows. First, many studies rely on the 80 or more features of CICFlowMeter and are therefore skewed toward TCP and UDP flow statistics. Second, most studies develop separate detection models specialized for a particular attack type (DDoS, PortScan, DoS, and so on), and research on frameworks that handle diverse attack types in an integrated way within a single pipeline is relatively scarce. Third, feature extraction and attack-interval segmentation are often performed by separate tools and are not integrated into an end-to-end pipeline. Fourth, many studies do not sufficiently combine application-layer features such as TLS, DNS, and HTTP with flow-level statistics.

To address these limitations, this study proposes a framework that implements (1) binary PCAP segmentation, (2) coverage of multiple attack types through five detection modes, and (3) extraction of 49 multi-protocol session features within a single integrated pipeline.

2.4. Summary of Related Frameworks

Table 1 compares the proposed platform with published research systems for packet-trace storage and retrieval, flow feature extraction and packet-level representation. Trace-storage systems such as Time Machine [12], pcapIndex [13] and NetStore [15] locate traffic of interest in large archives, but they keep the original trace and do not produce feature datasets. Flow analyzers such as CICFlowMeter [18,19], Joy [22], NFStream [28], Tranalyzer [29] and NTLFlowLyzer [30] produce per-flow feature records, and Joy, NFStream and Tranalyzer also expose application-layer metadata; none of them cuts attack intervals into separate PCAP files as a function of a detection result. NTLFlowLyzer, from the group that developed CICFlowMeter, computes 348 network- and transport-layer statistics per bidirectional flow and does not parse application-layer protocols. nPrint [31] represents each packet as a fixed-length bit vector for machine learning, which eliminates the need for manual feature design but also omits the protocol-level fields used here. The proposed platform combines rule-based attack-interval detection, byte-offset segmentation and a fixed 49-feature session schema in one pipeline; its detection is limited to the configurable rules described in Section 4.1.

Table 1. Comparison of the proposed platform with published research systems.

System	Primary Purpose	Attack-Interval PCAP Output	Session or Flow Features	TLS/DNS/HTTP Fields	Built-In Attack Detection
Time Machine [12]	Retrospective retrieval of recorded traffic	△ (query output; original trace kept)	—	—	— (triggered externally)
pcapIndex [13]	Index over unmodified trace files	△ (interval queries)	—	—	—
NetStore [15]	Storage and query of flow records	—	△ (flow records)	—	—
nDPI [21]	Deep packet inspection library	—	△ (protocol metadata)	✓	△ (flow risk flags)
Joy [22]	Flow and intraflow records (JSON)	—	✓	✓	—
CICFlowMeter [18,19]	Flow statistics for IDS datasets	—	✓ (80+)	—	—
NFStream [28]	Flow-based data analysis framework	—	✓	△ (via nDPI)	—
Tranalyzer [29]	Plugin-based flow analyzer	△ (per-flow packet export plugin)	✓	✓ (protocol plugins)	—

Table 1. *Cont.*

System	Primary Purpose	Attack-Interval PCAP Output	Session or Flow Features	TLS/DNS/HTTP Fields	Built-In Attack Detection
NTLFlowLyzer [30]	Network- and transport-layer flow features	—	✓ (348)	—	—
nPrint [31]	Bit-level packet representation for ML	—	△ (per-packet bits)	—	—
Proposed platform	Rule-based segmentation and feature extraction	✓ (byte-offset files)	✓ (49)	✓	✓ (five configurable rule modes)

✓: supported, △: partially supported or through an additional component, —: not supported.

3. System Architecture

This section describes the overall architecture of the proposed system and the interactions among its main modules. The system is implemented as a web application based on Spring Boot 2.6.13 and runs on Java 17. It parses binary PCAP files directly through the jNetPcap 1.4.r1425 native library and provides a web-based user interface through the Thymeleaf 3.0.15 template engine.

3.1. Comparison of Conventional Approaches with the Proposed Approach

Table 2 summarizes the functional differences between the proposed platform and existing related research platforms. Although existing tools excel at specific tasks, they do not support the entire process in an integrated manner—from attack detection to PCAP segmentation, session reconstruction, multi-protocol feature extraction, and structured dataset generation—within a single pipeline. Wireshark provides powerful capabilities for detailed packet inspection and decoding of various protocols, but it is not suitable for automated attack detection and session-level dataset generation from large-scale PCAP files. SplitCap has the advantage of efficiently splitting PCAP files, but its splitting criteria are primarily limited to session or flow units, and it does not provide detection or feature extraction functions based on attack events. Zeek excels in security monitoring and protocol-specific structured log generation, but separate scripts and post-processing steps are required to physically separate detected attack segments from the original PCAP at the byte offset level or to directly generate the fixed session-level feature vectors required in this study. CICFlowMeter has been widely used for generating machine learning-based intrusion detection datasets by providing numerous statistical flow features; however, it lacks PCAP attack segment detection and segmentation capabilities, and has limitations in comprehensively analyzing detailed information from the application layer—such as TLS, DNS, and HTTP—and payload entropy.

In contrast, the proposed framework integrates these individual functions into a single automated workflow. In particular, it can detect different types of network attacks according to user settings through five attack detection modes, and generate independent PCAP files by precisely separating detected attack segments based on byte offsets. Preserving the detected attack traffic itself, rather than merely logging its status, provides a reproducible unit of analysis. Furthermore, it automatically reconstructs bidirectional sessions for the separated PCAPs and generates 49 features in a consistent CSV format, including not only basic flow statistics but also TLS handshakes, DNS queries, HTTP metadata, payload entropy, and character distribution. Therefore, compared to existing approaches that require sequentially

combining separate PCAP segmentation, protocol analysis, and feature extraction tools, this approach can reduce the complexity of analysis procedures and data preprocessing. Another major advantage of our platform is that users can easily configure attack types or feature extraction settings via a web-based GUI.

Table 2. Comparison of the capabilities of existing tools and frameworks with those of the proposed system.

Capability	Wireshark [6]	SplitCap [5]	Zeek [7]	CICFlow Meter [18,19]	Proposed Framework
Primary purpose	Packet inspection	PCAP split	Security monitoring	Feature extraction	Unified analysis
Multi-type attack detection	—	—	△ (script)	—	✓ (5 modes)
Configurable detection params	—	—	△	—	✓
Byte-offset PCAP segmentation	—	—	—	—	✓
Hierarchical attack detection	—	—	—	—	✓
Session-level feature extraction	—	—	△ (logs)	✓ (80+)	✓ (49)
TLS handshake parsing	△ (manual)	—	△ (basic)	—	✓
DNS/HTTP deep parsing	△ (manual)	—	✓	—	✓
Shannon entropy analysis	—	—	—	—	✓
Structured CSV output	—	—	✓ (logs)	✓ (80+)	✓
Fully automated pipeline	—	—	△	△	✓
Constant memory (large files)	—	✓	—	—	✓ (chunked input)
Web-based visualization	✓	—	—	—	✓

✓: supported, △: partially supported or requiring additional development, —: not supported.

Furthermore, the proposed framework is designed for large-scale PCAP analysis. During the PCAP segmentation phase, byte offsets and zero-copy file transfer are used to process the entire input file without loading it into memory, and chunk-based input limits memory usage even when processing large volumes of traffic data. In addition, by providing automated feature extraction and SSE-based progress monitoring, it enhances operational convenience for long-duration network forensics and dataset generation tasks. Thus, the primary contribution of the proposed framework lies not in replacing the performance of specific single functions with existing individual tools, but in providing an end-to-end processing environment for network attack analysis by integrating attack detection, precise

PCAP segmentation, multi-protocol feature extraction, and analysis data generation into a single, consistent workflow.

3.2. Overview of the Pipeline

As shown in Figure 3, the proposed system follows a pipeline architecture consisting of four processing stages, of which Stages 2 and 3 (segmentation and feature extraction) are the core stages.

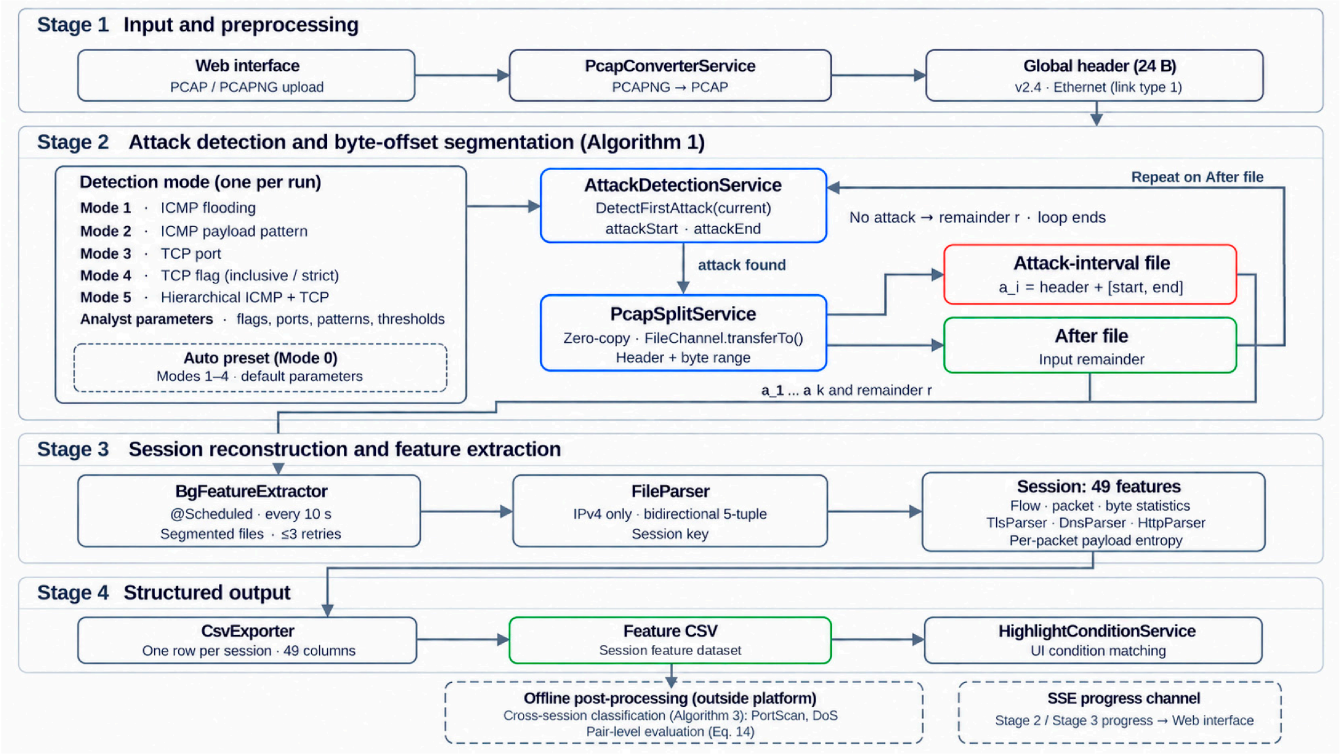


Figure 3. Processing pipeline of the platform. In the remainder-file variant shown, Stage 2 repeats detection on the remainder (After) file until no attack is found; the single-pass variant continues the scan in the original file (Section 3.3); cross-session classification (Section 4.3) runs offline on the feature CSV.

Stage 1: PCAP Input and Preprocessing. When a user uploads a PCAP file through the web interface, the system validates the file format. PcapConverterService converts various formats such as PCAPNG into standard PCAP (little-endian, magic 0xa1b2c3d4) to ensure consistency in subsequent processing. A 24-byte global header (version 2.4, snap length 65535, link-type 1 Ethernet) is written into the converted file.

Stage 2: Attack Detection and PCAP Segmentation. AttackDetectionService scans the PCAP file sequentially using the mode selected by the user from the five detection modes. The byte offsets of the interval judged to be an attack (attackStart, attackEnd) are recorded, and PcapSplitService segments the attack interval and the remaining interval into separate valid PCAP files using a zero-copy approach based on FileChannel.transferTo(). The detailed algorithms of the five detection modes are described in Section 4.

Stage 3: Session Reconstruction and Feature Extraction. FileParser parses every packet in the segmented PCAP file and reconstructs bidirectional sessions based on the 5-tuple. For each session, a Session object computes 49 multi-protocol features, and TlsParser, DnsParser, and HttpParser parse protocol fields according to RFC 9846, RFC 1035, and RFCs 9110 and 9112, respectively [32–35]. The detailed feature extraction methodology is described in Section 4.

Stage 4: Structured Output. CsvExporter writes the 49 features of each session in CSV form. The exported CSV file can be used as input to machine learning models after the column handling of Section 4.2.5. Internally, the system also determines whether each record matches the monitoring criteria defined by the highlight conditions (HighlightConditionService), which supports visual identification in the web interface.

3.3. Byte-Offset-Based Precise Segmentation

PCAP segmentation in the proposed system is performed at the byte-offset level rather than at the packet level. During jNetPcap's callback-based parsing, the absolute byte position of every packet within the file is tracked. Once the attack detection module determines the attack start offset (attackStart) and end offset (attackEnd), the FileChannel.transferTo() method of Java NIO transfers the corresponding byte range directly from the original file. Because transferTo() may transfer fewer bytes than requested [11], the call is repeated until the whole range has been written, and the size of each output file is checked against the requested range.

Each attack-interval file produced by segmentation is defined, as in Equation (1), as the concatenation of the 24-byte PCAP global header and the byte range of the detected attack interval.

$$F_{seg} = H_g \parallel F[o_s, o_e] \tag{1}$$

Here, H_g denotes the PCAP global header [36,37], and o_s and o_e denote the start and end byte offsets of the attack interval, respectively.

The key advantage of this approach is memory efficiency. Because the entire file is not loaded into memory and the zero-copy mechanism of the operating system is used instead [38,39], the segmentation module uses a constant amount of memory, approximately 30 MB (measured in-house) regardless of the file size. The 24-byte PCAP global header is automatically copied into each segmented file, which guarantees compatibility with standard tools such as Wireshark.

Accordingly, as expressed in Equation (2), the memory complexity of the segmentation module remains constant and independent of the input file size |F|, while Equation (3) extends the same accounting to end-to-end processing, in which the session state retained by the detection stage is added to that constant segmentation cost.

$$M_{split}(|F|) = O(1) \tag{2}$$

$$M_{e2e}(F) = M_{split} + \max_{c \in C(F)} O(|K(c)|) \tag{3}$$

Here, C(F) is the set of units (chunks) over which file F is scanned continuously, and K(c) is the set of unique session keys observed within chunk c. The memory usage of the segmentation stage is independent of the input size, but the session-tracking data structure of the detection stage is proportional to the number of unique flows within a single continuous scan range. End-to-end memory is therefore determined by the maximum rather than by the sum over all chunks, so that, in this model, chunk-wise input bounds the session state by the largest chunk rather than by the whole input.

$$T_{scan}(F) = \sum_{i=0}^{k} |F_i| = O(k \cdot |F|) \tag{4}$$

In Equation (4), F_i is the remaining file at the i-th iteration. When the segmented interval is sufficiently small relative to the remaining file, the total scan cost grows linearly with the number of segmentations k. For chunk-wise input, k per file is small and the effect is negligible, but for a single large file this term dominates the processing time. The single-pass variant reads the input once, so its scan cost is O(|F|) and it writes only the attack-interval files.

Algorithm 1 gives the two variants of the segmentation loop. In both, detection starts with empty state, and when an attack interval is found its byte range is copied into an attack-interval (Before) file and the detection state is cleared. The remainder-file variant, used in the first implementation and in the web interface, then writes the rest of the input as a remainder (After) file and restarts detection on it, so that every intermediate file is a valid PCAP. The single-pass variant continues with the packet that follows the interval in the original file. Because the remainder-file variant restarts with empty state exactly at attackEnd, both variants produce the same attack-interval files; this was verified on all intervals of the three evaluated datasets (Section 5.5.1). The single-pass variant uses the same splitting and feature-extraction classes and is run from a separate entry point.

Algorithm 1 Attack-Triggered Segmentation (Stage 2), single-pass and remainder-file variants

```
Input: original PCAP F, detection mode M, parameters theta, variant v in {single,
remainder}
Output: attack-segment files A = {a_1,...,a_k}
A <- empty;  S <- empty;  current <- F      // S: detection state of mode M
for each packet p of current, in file order, do
    (found, attackStart, attackEnd) <- Update(S, p, M, theta)    // Section 4.1.4 for
Mode 4
    if found then
        // zero-copy copy via FileChannel.transferTo(), repeated until complete,
Equation (2)
        a <- GLOBAL_HEADER(24B) || Bytes(current, attackStart, attackEnd)
        add a to A;  S <- empty
        if v = remainder then
            current <- GLOBAL_HEADER(24B) || Suffix(current, attackEnd)   //
After file
            restart the loop at the first packet of current
        end if   // v = single: continue with the packet after attackEnd
    end if
end for
return A
```

3.4. Automation and Instant Monitoring

The system provides two automation mechanisms. First, BgFeatureExtractor periodically scans the output directory through the @Scheduled(fixedDelay = 10 s) annotation, automatically discovers unprocessed PCAP files, and performs feature extraction. Only the attack-interval (Before) files are passed to feature extraction; the remainder (After) files created during iterative segmentation are intermediate products and are skipped. On failure, it retries up to three times, and it verifies the existence and size of the CSV file.

Second, the progress of long-running processing jobs is delivered to the client through the Server-Sent Events (SSE) protocol. The overall progress of the segmentation job and the feature extraction progress of individual files are managed on separate channels, so a user can check the current processing stage and progress in a web browser without refreshing the page.

4. Detection and Feature Extraction Methodology

This section describes the two core components of the proposed framework: the multi-mode attack detection algorithms and the session-level feature extraction methodology.

4.1. Multi-Mode Attack Detection

The proposed system provides five detection modes, each classified according to protocol type and detection strategy. In addition, the user interface offers an Auto preset (Mode 0) that applies Modes 1 to 4 in sequence with fixed default parameters (Section 5.1.1); it is a convenience option and not a separate detection method. All modes are combined with the byte-offset-based segmentation algorithm (Section 3.3) so that the detected attack interval is segmented precisely. The detection parameters of each mode (thresholds, flag combinations, target ports, and so on) are user-configurable, which allows the same detection mode to cover diverse attack types without changing the implementation. The evaluation in Section 5 uses Mode 4 only; Modes 1, 2, 3 and 5 are described as implemented and were not evaluated in this paper.

4.1.1. ICMP Flooding Detection (Mode 1)

Mode 1 detects flooding attacks based on the frequency of ICMP packets. While scanning the PCAP file sequentially, it counts the number of ICMP Echo packets [40] occurring within a fixed one-second time window, and when that count reaches the threshold (four packets), the interval is judged to be an attack. The attack start offset is set to the position of the first ICMP packet at the moment the threshold is exceeded, and the attack end offset is set to the end position of the packet at which the number of additional ICMP Echo packets observed after the attack start reaches the termination threshold (four packets).

This mode is designed to detect traditional ICMP flooding attacks that exhaust bandwidth by transmitting a large volume of ICMP Echo Requests and Replies in a short period.

4.1.2. ICMP Payload Pattern Detection (Mode 2)

Mode 2 uses user-specified payload patterns in ICMP Echo packets to delimit an attack interval. When an ICMP Echo packet containing the user-specified start pattern is observed through PatternMatcher, its position is recorded as the attack start offset, and when an Echo packet containing the end pattern is subsequently observed, its position is taken as the attack end offset. Unlike frequency-based Mode 1, Mode 2 therefore uses a pair of start and end payload patterns as interval delimiters.

Because the start and end patterns are supplied by the user, this mode can also isolate traffic carrying a distinctive payload signature, such as a marker deliberately injected into controlled test traffic or a fixed payload explicitly configured in a packet-crafting tool such as hping3.

4.1.3. TCP Port-Based Detection (Mode 3)

Mode 3 detects TCP-based attacks targeting a specific port. When TCP traffic whose source port equals the user-specified service port (targetPort) is observed at or above the threshold (pattern_count), the interval is judged to be an attack. This mode is used to detect concentrated attacks on known vulnerable ports or denial-of-service attempts.

4.1.4. TCP Flag Analysis Detection (Mode 4)

Mode 4 detects abnormal flag transition patterns within TCP sessions. When a user specifies a start flag (for example, SYN) and an end flag (for example, RST), the system tracks TCP sessions using a bidirectional session key in which the IP address and port pairs are normalized in lexicographic order. It judges the interval to be an attack when the pattern of a session initiated by the specified start flag and completed by the end flag occurs at least pattern_count times.

Flag matching is performed using a bitmask. Because every combination containing the specified flags is matched, if the start flag is specified as SYN, for example, not only pure

SYN packets but all packets with the SYN bit set, such as SYN + ACK and SYN + FIN, are matched. Such inclusive matching can counter attempts to evade detection by combining flags. When normal connection termination (FIN) and forced abnormal termination (RST) must be strictly distinguished, however, strict matching, which matches only packets whose flags exactly equal the specified flags, can be applied as an option. Strict matching is applied to the start flag that determines session initiation, whereas the end flag is always evaluated with inclusive matching so that variants carrying additional flags in the response packet (for example, RST + ACK) are covered.

$$k(p) = (min(e_s,e_d),max(e_s,e_d),proto) \tag{5}$$

In Equation (5), e_s and e_d are the (IP, port) pairs of the source and the destination, respectively. Lexicographic sorting normalizes both directions of the same session into a single key.

The flag matching conditions are defined by inclusive matching in Equation (6) and strict matching in Equation (7).

$$match(f, m) = ((f \wedge m) = m) \tag{6}$$

$$match_{strict}(f, m) = (f = m) \tag{7}$$

Here, f is the TCP flag field of the packet and m is the flag mask specified by the user [41]. Inclusive matching (Equation (6)) matches every flag combination that contains the specified flags, whereas strict matching (Equation (7)) matches only packets whose flags exactly equal the specified flags and thereby excludes packets carrying additional flags such as SYN + ACK.

The segmentation trigger condition is defined as follows. For a packet sequence P = <p_1, ..., p_n> and key k, the most recent position at which the start flag matched and the number of end-flag matches after that position are given by Equation (8) and Equation (9), respectively.

$$s_k(j) = max\{i \le j : k(p_i) = k \wedge match_{strict}(f_i,m_s)\} \tag{8}$$

$$c_k(j) = |\{i : s_k(j) < i \le j, k(p_i) = k \wedge match(f_i,m_e)\}| \tag{9}$$

Segmentation occurs at the first j that satisfies the condition of Equation (10), and the segmented interval is [off(p_{s_k(j)}), off(p_j) + ℓ(p_j)).

$$c_k(j) \ge \tau \ \wedge \ off(p_j) + \ell(p_j) - off(p_{s_k(j)}) \ge L_{min} \tag{10}$$

Here, τ is pattern_count and L_min is the minimum interval length (1000 bytes). Equation (8) is written for the strict option; when strict matching is disabled, match_strict is replaced by the inclusive match of Equation (6), as in line 8 of Algorithm 2. Because the end counter is reset when the start flag matches again, segmentation occurs only when the number of end flags observed consecutively without an intervening start flag for the same key is at least τ. Combined with the fact that the session key includes the port, this condition implies that traffic in which only a single SYN-RST pair occurs per destination port cannot satisfy $\tau \ge 2$.

In Mode 4, adjusting the flag combination and pattern_count allows different attack types to be targeted, such as SYN flood (SYN to RST with a high pattern_count), port scan (SYN to RST with a low pattern_count together with destination port diversity), and abnormal connection termination (SYN to FIN).

Algorithm 2 TCP-Flag Session Detection (Mode 4) with Inclusive/Strict Matching

```
Input: packet stream P, start mask m_s, end mask m_e,
pattern_count tau, strict in {true, false}, minimum length L_min
Output: attack byte range (attackStart, attackEnd) or NULL
match(f,m)       := ((f AND m) = m)       // Equation (6) inclusive
match_strict(f,m) := (f = m)              // Equation (7) exact
sessions <- empty map       // key: bidirectional normalized key
// sessions[k].endCnt: end-flag count; end(p) = p.offset + len(p)
for each packet p in P do
    k <- BiKey(p)             // lexicographically normalized (ip,port)
    f <- p.tcp_flags
    if (strict ? match_strict(f,m_s) : match(f,m_s)) then
        sessions[k].opened   <- true;  sessions[k].endCnt <- 0
        sessions[k].startOff <- p.offset
    end if
    if match(f,m_e) and sessions[k].opened then   // end: always inclusive
        sessions[k].endCnt <- sessions[k].endCnt + 1
    end if
    if sessions[k].endCnt >= tau and end(p) - sessions[k].startOff >= L_min then
        return (sessions[k].startOff, end(p))   // Equation (10)
    end if
end for
return NULL
```

4.1.5. Hierarchical Detection (Mode 5)

Mode 5 is a hierarchical detection scheme that simultaneously detects lower-layer attack patterns inside an upper-layer attack interval. Detection is carried out in a single scan over the original PCAP. When the start of an attack interval is identified during the scan according to an upper-layer condition (for example, the ICMP flooding of Mode 1), TCP sessions inside that interval are tracked concurrently during that interval, and any session satisfying the lower-layer condition (the specified flag pattern identical to that of Mode 4 occurring at least pattern_count times) is judged to be a lower-layer attack and its byte interval is recorded. Once the end of the upper interval is confirmed, the upper interval is segmented from the original file as an attack-interval file (Before file), and each recorded lower-layer session interval is segmented directly from the original file as a separate hierarchy file.

Mode 5 operates in two variants depending on how the upper-layer condition is defined, and both variants share the same layer structure with ICMP as the upper layer and TCP as the lower layer. The basic variant (Single) identifies the upper ICMP interval using the same flooding-frequency condition as Mode 1. The lower TCP sessions of the Single variant are tracked with a unidirectional key based on the 5-tuple, which prevents mismatches with normal termination packets from the server side by matching only the end flags observed in the direction that initiated the session. In addition, sessions whose elapsed time between start and end exceeds a threshold (one second) are regarded as normal TCP connections and are excluded from lower-layer attacks. The extended variant (Mixed) defines the boundaries of the upper ICMP interval by start and end payload pattern pings as in Mode 2, tracks lower TCP sessions with the same bidirectional normalized key as Mode 4, and additionally supports scenarios in which the lower TCP attack starts before the upper ICMP interval (the splitOrder setting). Users switch between the two variants solely through parameter settings.

This hierarchical approach makes it possible to capture, within a single analysis pass, composite attack scenarios in which bandwidth is exhausted by ICMP flooding while TCP-based attacks (port scanning, SYN flooding, and so on) occur simultaneously.

4.1.6. Parameter Settings by Attack Type

The detection modes are rule templates whose parameters are set by the analyst for the attack under study. Table 3 summarizes the detection mode and parameter settings used for each attack type in this study.

Table 3. Detection modes and parameter settings by attack type.

Attack Type	Detection Mode	Start Flag	End Flag	pattern_count	Remarks
DDoS (SYN Flood)	Mode 4	SYN	RST	High (≥4)	Direct detection in Stage 2
PortScan	Mode 4	SYN	RST	Low (≥2)	Distinguished by destination port diversity
DoS (HTTP Flood)	Mode 4	SYN	RST	High (≥4)	Distinguished by session_count
ICMP Flooding	Mode 1	—	—	—	icmpCount threshold
Composite attack	Mode 5	Configurable	Configurable	Configurable	Hierarchical combination

As shown in this table, DDoS and PortScan use the same SYN to RST flag combination but are distinguished by the auxiliary indicators of pattern_count and destination port diversity. Whether other attack types can be covered by parameter changes alone was not evaluated in this study.

4.2. Session-Level Feature Extraction Framework

The attack-interval PCAP files segmented in Stage 2 become the input to session-level feature extraction in Stage 3. This section describes the session reconstruction method and the extraction procedure for the 49 features.

4.2.1. Session Reconstruction

FileParser parses every packet in the segmented PCAP file sequentially and reconstructs bidirectional sessions based on the 5-tuple (source IP, source port, destination IP, destination port, protocol). The bidirectional session key normalizes source and destination so that (A to B) and (B to A) traffic are merged into the same session. Whenever a packet arrives, the statistics of the corresponding session (packet count, byte count, timestamps, and so on) are updated instantly. The forward direction of a session is the direction of its first observed packet. Only IPv4 packets are parsed. By default, no termination rule or idle timeout is applied, so packets that share a 5-tuple within one file form a single session.

Three session notions are used in this paper. (i) A Stage 2 flag session is the detection state kept for a key k of Equation (5): the offset of the latest start-flag match and the number of end-flag matches after it (Equations (8) and (9)). It exists only during detection and is reset when the start flag matches again. (ii) A Stage 3 session record for key k in a file F is, by default, the set of all packets p in F with $k(p) = k$; the 49 features of Table 4 are computed per record. (iii) A CICFlowMeter flow is the unit of the published CICIDS-2017 labels: a bidirectional flow that CICFlowMeter ends at a FIN packet or after a flow timeout (120 s by default). It is used only as ground truth (Section 5.1.3).

Table 4. The 49 features classified by category.

Category	No. of Features	Representative Features
Identification	5	source_ip, source_port, destination_ip, destination_port, protocol
Temporal statistics	4	flow_start_time, flow_end_time, flow_iat_list, flow_iat_std
Packet statistics	6	fwd/bwd/total_packet_count, fwd/bwd/payload_packet_count
Byte statistics	5	fwd_total_bytes, bwd_total_bytes, payload_length_list/max/min
TLS handshake	14	tls_version, ciphersuite, extension, certificate, etc.
DNS query	7	dns_query_length, dns_query_suffix, TTL, character distribution, etc.
HTTP metadata	3	content_type, user_agent, response_code
Payload analysis	5	payload_entropy_list, payload_digit/alpha/special/nonascii_count
Total	49	

Stage 3 also offers an optional termination rule for (ii). A record is closed after an RST packet, after FIN has been seen in both directions (one trailing pure ACK is still added), or when no packet with the key arrived for an idle time T_idle; the next packet with the same key starts a new record. A closed record is summarized and released, so memory holds only open records. With T_idle = 120 s on the full Friday and Wednesday files, this produced 582,096 and 721,250 records instead of 396,968 and 226,725, against 695,257 and 777,037 CICFlowMeter flows (Section 5.6.2). The pipeline results in Section 5 use the default setting.

The session key of the detection stage (Stage 2) described in Section 4.1, however, is designed to meet the detection requirements of each mode. TCP flag-based detection (Mode 4) must associate a SYN in the attacker-to-victim direction with an RST response in the victim-to-attacker direction as the same session, so it uses the same bidirectional normalized key as the feature extraction stage. In contrast, port-based detection (Mode 3) and the lower-layer TCP tracking of hierarchical detection (the Single variant of Mode 5) retain a unidirectional key to prevent mismatches with normal termination packets from the server side. This per-mode session key design reflects the directional requirements of the detection logic in each mode and the requirement of the feature extraction stage to merge bidirectional traffic, respectively.

4.2.2. Basic Network Features

The basic network features extracted for every session are as follows.

Identification features: source_ip, source_port, destination_ip, destination_port, and protocol, which form the 5-tuple that uniquely identifies a session.

Temporal features: flow_start_time, flow_end_time, flow_iat_list, and flow_iat_std, which include the start and end times of the session, the list of inter-arrival times between packets, and the standard deviation of the inter-arrival times. In particular, flow_iat_list and flow_iat_std are effective features for distinguishing patterns in normal traffic from those in attack traffic.

Packet statistics features: fwd_packet_count, bwd_packet_count, total_packet_count, fwd_payload_packet_count, bwd_payload_packet_count, and payload_packet_count, which aggregate the numbers of forward and backward packets and the numbers of packets carrying a payload. DDoS attacks show an abnormally high forward packet count, whereas PortScan shows similar forward and backward packet counts.

Byte statistics features: fwd_total_bytes, bwd_total_bytes, payload_length_list, payload_length_max, and payload_length_min, which provide the total number of transmitted bytes and statistics on payload length.

4.2.3. Per-Protocol Deep Parsing

In addition to the basic network features, the proposed system performs deep parsing of the TLS, DNS, and HTTP protocols to extract protocol-specific features.

TLS features. TlsParser parses the ClientHello and ServerHello messages of the TLS handshake and extracts the following features: tls_version_hex (protocol version code), tls_version (human-readable version name), client_ciphersuite (cipher suites proposed by the client), server_ciphersuite (cipher suite selected by the server), client_extensions and client_extension_count (list and number of client extensions), server_extensions and server_extension_count (list and number of server extensions), tls_supported_groups (supported groups), tls_ec_point_format (elliptic curve point format), public_key_length (public key length), server_certificate_count (number of server certificates), certificate_validity (certificate validity period), and tls_san_count (number of Subject Alternative Names).

These TLS metadata features can support classification of benign and malicious encrypted traffic, including command and control (C2) communication [42–44]. Identifying malicious certificates specifically requires certificate-focused validation and is not claimed here.

DNS features. DnsParser parses DNS query and response messages and extracts the following features: dns_query_length (length of the DNS message), dns_query_suffix (query domain suffix), dns_response_ttl_list (response TTL values), dns_query_digit_count (number of digit characters in the query), dns_query_alpha_count (number of alphabetic characters), dns_query_special_count (number of special characters), and dns_query_nonascii_count (number of non-ASCII characters).

Character-level and entropy features of DNS names have been used for DNS tunneling detection [45] and for the analysis of domains produced by a domain generation algorithm (DGA) [46]. DGA domains in particular exhibit a higher proportion of digits and special characters and higher entropy than ordinary domains.

HTTP features. HttpParser parses HTTP request and response messages and extracts the following features: http_content_type (content type), http_user_agent (user agent), and http_response_code (response code). An unusual User-Agent string or a non-standard Content-Type can be an indicator of a web attack or malicious bot activity.

4.2.4. Payload Entropy and Character Distribution

As protocol-independent features, Shannon entropy and character distribution are computed over the payload of each packet in a session and stored as lists in packet arrival order; packets without a payload contribute 0.

The Shannon entropy of payload P is defined as in Equation (11) [47], and its per-packet values form the session feature payload_entropy_list.

$$H(P) = -\sum_{x=0}^{255} p(x)\, log_2\, p(x) \tag{11}$$

where p(x) is the relative frequency of byte value x among the bytes that constitute the payload. H(P) takes values from 0 (a payload consisting of a single repeated byte value) to 8 (a uniform, completely random byte distribution), and a larger value indicates greater payload randomness.

payload_entropy_list: the list of per-packet Shannon entropy values of the payload, each ranging from 0 (a single repeated byte value) to 8 (a uniform, completely random byte distribution). Encrypted services tend to exhibit entropy values closer to the upper end of the 0 to 8 range, whereas plaintext services generally show lower values; the operating threshold is protocol- and dataset-dependent [48].

payload_digit_count, payload_alpha_count, payload_special_count, and payload_nonascii_count: the numbers of digits, alphabetic characters, special characters, and non-

ASCII characters in the payload, respectively. These character distribution features are used to determine whether traffic is encrypted, to identify binary protocols, and to detect encoded payloads.

4.2.5. Summary of the 49 Features

Table 4 summarizes the 49 features by category, together with the number of features in each category and representative examples.

Section 5.6.4 uses the features in a classification experiment with the following column handling. The four endpoint identifiers (source and destination address and port) and the two absolute timestamps (flow_start_time and flow_end_time) are not used as model inputs; the protocol number is kept as a code. List-valued columns with one value per packet (for example, payload_lens, entropys and flow_delta_times) are summarized by their mean, standard deviation, minimum and maximum. Lists of TLS code points (cipher suites, extensions, groups and point formats) are summarized by their length, hexadecimal scalars (TLS version and server cipher suite) are converted to integers, certificate validity to days, and free-text columns (HTTP user agent, HTTP content type and DNS query suffix) to a presence flag. This yields 79 numeric model inputs. Unlike the more than 80 features of CICFlowMeter, which focus mainly on TCP and UDP flow-level statistics (packet size distribution, IAT statistics, and so on), the proposed framework provides application-layer behavioral features in an integrated manner, including deep parsing of the TLS, DNS, and HTTP protocols and payload entropy analysis.

4.3. Attack Type Classification Based on Session Features

Once suspected attack intervals are segmented in Stage 2 and the 49 features are extracted in Stage 3, the attack type can be identified from these features [49]. For attack types that cannot be distinguished by the flag pattern of a single session, this study defines a secondary classification rule (Table 5) that exploits the statistical properties of a set of sessions. This secondary classification is not a capability implemented in the framework itself but a separate post-processing analysis procedure that takes the session feature CSV produced in Stage 4 as its input.

Table 5. Session feature profiles by attack type.

Attack Type	Key Characteristic	Classification Condition
DDoS	High-frequency SYN→RST repetition within a single session	Direct detection in Stage 2 (Mode 4, pc ≥ 4)
PortScan	Many unique dst_port values per (src,dst) pair, ≤5 packets, no payload	protocol = TCP, total_packet_count ≤ 5, payload_packet_count = 0, unique_dst_ports ≥ 50
DoS (HTTP)	Extremely large number of sessions per (src,dst,port) triple	protocol = TCP, dst_port ∈ {80,443,8080}, session_count ≥ 10,000

4.3.1. Session Profiles by Attack Type

Each attack type exhibits a distinctive profile in the session feature space. Table 5 summarizes the feature profiles of the three representative attacks validated in this study.

4.3.2. Two-Stage Detection Procedure

Stage 2 segmentation and feature-based post-processing classification complement each other. Stage 2 performs first-pass filtering of high-volume attack traffic using patterns observable within a single session, such as TCP flag sequences, ICMP Echo patterns, and payload signatures. Post-processing classification takes the session feature CSV extracted

by the framework as input and either refines the attack type or identifies attacks that are indistinguishable from normal traffic at the single-session level (for example, PortScan) through set-level statistics.

This separation of layers offers the following design advantages. First, attacks with a clear flag pattern such as DDoS can be segmented and labeled immediately in Stage 2 without overhead. Second, attacks such as PortScan, whose individual sessions are indistinguishable from normal traffic but which are revealed at the set level, can also be handled within the same framework. Third, even when a new attack type is added, the detection coverage could in principle be extended by adding profile definitions of the kind shown in Table 5 without redesigning the framework; this was not evaluated in this study. Algorithm 3 gives the cross-session rules used for PortScan and DoS in this study.

Algorithm 3 Cross-Session Attack Classification

```
Input: bidirectional sessions S (49 features each), profiles in Table 5
Output: detected attack pairs/triples D
D <- empty
// ---- PortScan : destination-port diversity per (src,dst) ----
for each canonical pair (a,b) in S do      // direction normalization: (A,B),(B,A) -> single pair
    cand <- { s in S(a,b) : s.proto = TCP          // includes bidirectional traffic
                                  and s.total_packet_count   <= 5
                                  and s.payload_packet_count = 0 }
    U <- count of distinct s.dport for s in cand
    if U >= UNIQUE_DST_PORTS_MIN then              // = 50
        add (a,b,"PortScan") to D
    end if
end for
// ---- DoS : session concentration per (src,dst,dport) ----
for each (src,dst,dport) triple in S do
    if proto = TCP and dport in {80,443,8080} then
        c <- session_count(src,dst,dport)
        if c >= SESSION_COUNT_MIN then             // = 10,000
            add (src,dst,dport,"DoS") to D
        end if
    end if
end for
return D
```

To validate the detection modes, feature extraction, and per-attack-type classification rules proposed in Section 4, Section 5 presents experimental results for the three attack types of DDoS, PortScan, and DoS using the CICDDoS2019 and CICIDS-2017 datasets.

5. Experiments and Evaluation

This section presents functional validation confirming that the pipeline behaves as designed, together with performance evaluation results that quantify its processing cost. Detection experiments were conducted for DDoS, PortScan, and DoS to confirm that all three attack types can be handled by the same detection mode with different parameter settings. Whether the feature set generalizes to unseen datasets or attack types is a question that requires a separate validation protocol and is outside the scope of this paper.

5.1. Experimental Setup

5.1.1. Experimental Environment and User Interface

All experiments were conducted on Windows 11. The hardware consisted of an Intel Core i9-14900 CPU with 32 logical cores and 64 GB of RAM. The software environment used JDK 17.0.12+8-LTS-286 with the G1 garbage collector, launched with -Xms512m -Xmx<N> -XX:TieredStopAtLevel=1, where the heap ceiling <N> was varied per experiment as reported in Table 6. Heap usage, non-heap usage, direct buffers, and garbage-collection time were sampled at one-second intervals, and process resident set size was collected at the same interval from a separate process. All inputs and outputs of the measurements in Section 5 were on the internal disk of this machine.

Table 6. System performance metrics (Detected Attacks = number of attack-interval files produced by segmentation).

Metric	CICDDoS2019 (DDoS)	CICIDS-2017 Fri (PortScan)	CICIDS-2017 Wed (DoS)
File Size	179.44 GiB (965 chunks)	8.07 GiB (single file)	12.28 GiB (single file)
Processing Time (remainder-file variant)	1402 s (23.4 min)	5628 s (1.56 h)	5603 s (1.56 h)
Heap Peak	heap used peak 1225 MB	heap used peak 853 MB	heap used peak 906 MB
RSS Peak	RSS peak 1726 MB	RSS peak 1502 MB	RSS peak 1343 MB
Minimum Heap	Min heap 1 GB	Min heap 2 GB	Min heap 2 GB
CPU Average	3.68%	2.42%	2.47%
CPU Maximum	12.35%	8.82%	7.85%
Detected Attacks	1485	2157	1760
Avg Time per Attack	0.94 s	2.61 s	3.18 s
Processing Time (single pass)	706 s (11.8 min)	363 s (6.1 min)	369 s (6.1 min)
RSS Peak (single pass)	1595 MB	1088 MB	1148 MB
Bytes Written (remainder-file/single pass)	120.4 GiB/3.08 GiB	6019 GiB/1.91 GiB	5856 GiB/2.54 GiB
Peak Temporary Disk (remainder-file variant)	3.41 GiB	16.1 GiB	24.5 GiB

Note: All experiments were run on the platform described in Section 5.1.1 (Intel Core i9-14900, 32 logical cores, 64 GB RAM; JDK 17.0.12+8-LTS-286, G1 garbage collector, -Xms512m -Xmx<N> -XX:TieredStopAtLevel=1). CPU figures are the process CPU load normalized over all 32 logical cores, not over a single core. RSS Peak is the maximum of the one-second samples. Each column reports a single run for all rows except Minimum Heap, which is the lowest heap ceiling at which a run of the same input completed; the CICDDoS2019 column is the run that produced the counts of Table 7. All three columns were run with a heap ceiling of 4 GB; the lowest ceilings tried were 1 GB for CICDDoS2019 and 2 GB for CICIDS-2017. All inputs and outputs were on the internal disk; the CICDDoS2019 runs were repeated from it with the same detection counts. The single-pass rows were measured with a separate entry point that runs the Mode 4 rule of Algorithm 2 in one pass and Stage 3 on each attack-interval file right after it is written, sequentially and without the Spring scheduler, at a 4 GB heap ceiling. It produced the same attack-interval byte ranges and the same Stage 3 session counts as the remainder-file runs. Bytes written count all PCAP files written by Stage 2; the peak temporary disk counts the attack-interval files and the remainder files present at the same time.

The framework itself runs on Java 17, Spring Boot 2.6.13, and jNetPcap 1.4.r1425. Session feature extraction is performed as a background batch job based on Spring's @Scheduled scheduler.

The platform supports both automatic and user-specified attack detection modes. In the Auto preset (Mode 0), the platform applies Mode 1, Mode 2, Mode 3 and Mode 4 (SYN to FIN) to the same input in sequence, each with fixed default parameters, and reports the attack intervals found by each mode; it does not infer the attack type from the input. Alternatively, users can explicitly select a detection method or condition when the

target attack characteristics are known. The first PCAP file (Friday-WorkingHours.pcap) is configured to operate in Auto mode, whereas the second PCAP file (running_test.pcap) is configured to use a TCP Flag-based detection mode. In the latter case, TCP flag conditions, such as SYN and RST, can be specified to identify and segment traffic associated with particular attack behaviors. All experiments in Section 5 were run with an explicitly selected mode and the parameters given in each subsection, not with Auto.

Table 7. Experimental results summary.

Attack Type	Dataset	Pair Precision	Pair Recall	Session Precision	Session Recall
DDoS	CICDDoS2019	100% (2/2)	100% (2/2)	99.52% (2,757,757/2,771,186)	—
PortScan	CICIDS-2017 Fri	100% (1/1)	100% (1/1)	100% (20,821/20,821)	13.1%
DoS	CICIDS-2017 Wed	100% (1/1)	100% (1/1)	100% (167,305/167,305)	66.2%

Note: Session Precision for PortScan and DoS is computed over the sessions classified as attacks by the cross-session rules of Algorithm 3 (denominators 20,821 and 167,305), whereas for DDoS it is computed over all sessions extracted by Stage 3 (denominator 2,771,186), since DDoS is detected directly in Stage 2. Evaluated on the Stage 3 denominator instead, PortScan and DoS yield 21.05% (36,077/171,405) and 75.88% (238,467/314,252), respectively. Session Recall denominators are the published CICIDS-2017 label counts. Session Recall is not reported for DDoS because the CICDDoS2019 ground truth is provided in CICFlowMeter timeout-flow units, which are not commensurable with the SYN-to-RST session definition used here; attacker-level completeness is instead reported as pair-level recall. Flow-level recall against the published CICIDS-2017 labels (Section 5.1.3) was 7.5% for PortScan and 91.1% for DoS.

After the user initiates processing by clicking the Start split button, the platform performs packet parsing, traffic analysis, attack detection, and flow/session-level segmentation and feature extraction. During this process, execution status and diagnostic information are continuously displayed in the Processing Console located at the bottom of the interface. The console automatically scrolls as new messages are generated, allowing users to monitor intermediate processing results, packet parsing events, segmentation operations, and other runtime information.

The fast-logging functionality is implemented using Server-Sent Events (SSE). SSE is a web-standard technology that enables unidirectional, fast data transmission from a server to a client over a persistent HTTP connection. In the proposed platform, SSE is used to continuously deliver processing and log messages generated by the backend packet-analysis engine to the web-based user interface. This mechanism allows users to monitor the progress of computationally intensive PCAP parsing and session-level analysis without repeatedly refreshing the page or issuing additional polling requests.

Once analysis of the network traffic captured in PCAP format begins, the system continuously generates and presents statistical information obtained during packet parsing, flow/session extraction, and attack detection. This dashboard enables users to monitor the characteristics of the extracted flows and sessions as well as the occurrence of attack patterns detected during the analysis process.

For each extracted flow or session, the platform provides statistical information on the observed traffic features and the corresponding attack detection results. In particular, the interface visualizes how frequently the predefined detection conditions are matched within individual flows or sessions, allowing users to examine the distribution and occurrence of attack-related characteristics across the analyzed network traffic. Summary statistics, including the number of matched instances, the total number of analyzed records, and the average matching rate, are presented together with graphical representations to facilitate intuitive interpretation of the analysis results.

Since PCAP processing and flow/session-level analysis are performed continuously, the statistical information displayed on the dashboard is dynamically updated throughout

the analysis process. Accordingly, both the numerical indicators and the graphical elements shown in the interface change instantly as additional packets are parsed and new flows or sessions are extracted and analyzed. This dynamic visualization provides users with continuous visibility into the progress of the analysis and enables them to observe how attack-related patterns emerge and evolve across the processed network traffic.

5.1.2. Datasets

Two public datasets were used for the evaluation because they provide original PCAPs and documented attack scenarios, and a third was used only for a transfer check. Constructing intrusion detection datasets from realistic traffic remains an active research problem [50], and using these two widely cited corpora ensured both diversity in attack types and experimental reproducibility.

(1) CICDDoS2019 [51]: a DDoS-specific dataset released by the Canadian Institute for Cybersecurity (CIC) that includes diverse DDoS variants such as SYN flood, UDP flood, and NTP amplification. In these experiments, the original PCAP files of both capture days were used (file prefixes 01-12 and 03-11, 819 and 146 files, 179.44 GiB in total); both days contain SYN flood traffic.

(2) CICIDS-2017 [20]: a comprehensive intrusion detection dataset released by CIC that simulates a real network environment in which normal traffic and attack traffic are mixed over five days. This study used the following two files:

- Friday-WorkingHours.pcap: contains a PortScan attack (attacker 172.16.0.1, victim 192.168.10.50).
- Wednesday-workingHours.pcap: contains four types of DoS attacks, namely Hulk, GoldenEye, Slowloris, and SlowHTTPTest (same attacker and victim pair).

Both files are distributed in the pcapng format despite the .pcap extension. They were converted once to the classic libpcap format (8.07 GiB and 12.28 GiB), and the converted files were used as input in all experiments, including the benchmark in Section 5.6.2.

(3) CSE-CIC-IDS2018 [52]: used only to check whether the DoS settings of Section 5.4 carry over to another corpus without retuning (Section 5.6.5). The captures of the victim web server (172.31.69.25) on 15 February 2018 (DoS-GoldenEye and DoS-Slowloris) and 16 February 2018 (DoS-SlowHTTPTest and DoS-Hulk) were used; the 16 February capture was converted from pcapng to the classic format before processing.

5.1.3. Evaluation Metrics

Detection performance was evaluated with the precision of Equation (12) and the recall of Equation (13).

$$Precision = \frac{TP}{TP + FP} \tag{12}$$

$$Recall = \frac{TP}{TP + FN} \tag{13}$$

$$\pi(a,b) = (min(a,b),max(a,b)),\ TP_{pair} = |\{\pi(a,b) : (a,b) \in D\} \cap \Pi_{GT}| \tag{14}$$

Pair-level matching is normalized as in Equation (14). As in session key normalization, source and destination are sorted lexicographically so that both directions are aggregated into a single pair. Equation (14) takes as input the set D of pairs judged to be attacks by the classification rules and the set Π_GT of normalized ground-truth attack pairs, and outputs the set of matched pairs used in Equations (12) and (13) at the pair level; it is applied only in the evaluation and does not change any state of the detection pipeline.

$$D_{scan} = \{(a,b) : |\{port_d(s) : s \in S(a,b), tcp(s), n_{pkt}(s) \leq 5, n_{pay}(s) = 0\}| \geq \theta_p\} \tag{15}$$

$$D_{dos} = \{(a,b,q) : q \in \{80,443,8080\}, |\{s \in S(a,b) : tcp(s), port_d(s) = q\}| \geq \theta_s\} \quad (16)$$

The cross-session classification rules for PortScan and DoS are given by Equation (15) and Equation (16), respectively, where S(a,b) is the set of sessions belonging to pair (a,b) and θ_p and θ_s are the thresholds for destination port diversity and session count. This paper uses θ_p = 50 and θ_s = 10,000, which are operating points for these datasets and were not derived from separate validation data.

Here, TP (true positive) is the number of actual attacks detected as attacks, FP (false positive) is the number of normal cases falsely detected as attacks, and FN (false negative) is the number of actual attacks that were missed. This study measures precision at two levels. Pair-level precision denotes the proportion of actual attack pairs among the detected (source_ip, destination_ip) pairs, and session-level precision denotes the proportion of actual attack sessions among the individually detected sessions.

Ground truth was assigned by endpoint pair. A session is labelled as an attack when its two endpoints are the published attacker and victim of the capture (CICDDoS2019: 172.16.0.5 with 192.168.50.1 or 192.168.50.4; CICIDS-2017: 172.16.0.1 with 192.168.10.50); this labelling gives the session precision and recall of Table 7. Because the published CICIDS-2017 labels are CICFlowMeter flows, a flow-level recall was also computed. Each labelled flow was mapped to the Stage 3 sessions with the same normalized 5-tuple whose time span, widened by 120 s, contains the flow start (the published timestamps have a resolution of one minute), and a labelled attack flow counts as recalled when it maps to a session classified as an attack. This mapping is approximate: 22.5% and 35.3% of the Friday and Wednesday sessions have no labelled flow under it.

5.2. DDoS Attack Detection Experiment (CICDDoS2019)

This section describes the results of a DDoS attack detection experiment conducted on the CICDDoS2019 dataset. The data used were the original PCAP files of both CICDDoS2019 capture days (965 files, 179 GiB), which contain SYN flood traffic. The detection configuration applied Mode 4 (TCP flag), used the SYN to RST pattern as the TCP flag condition, and set the minimum packet count (pc) threshold to 4. In the experimental environment, the attacker IP was identified as 172.16.0.5 and the victim IPs as 192.168.50.1 and 192.168.50.4. The large PCAP files were processed sequentially through byte-offset-based iterative segmentation. Start-flag matching was performed with inclusive matching (strict matching disabled).

As a result, ground-truth matching over all 1485 segmented files yielded a session-level precision of 99.52% (2,757,757/2,771,186) according to Equation (12). The 13,429 normal sessions (0.48%) included as false positives corresponded to background traffic extracted together with the attack interval at segmentation boundaries. By destination port, HTTPS (31.5%), DNS (27.1%), and HTTP (15.3%) accounted for 73.9% of the total, and by protocol the distribution was TCP 65.2%, UDP 29.8%, and OSPF 5.1%. The fact that about 30% of the false positives are UDP sessions supports the interpretation that they are background traffic included in the segmented interval irrespective of the SYN to RST flag transition. The main cause of the false positives lies in the structure of the present system, which is designed to preserve forensic context and therefore deliberately does not perform additional filtering beyond the TCP flag pattern in Stage 2. Figure 4 shows the destination port distribution of these false-positive sessions.

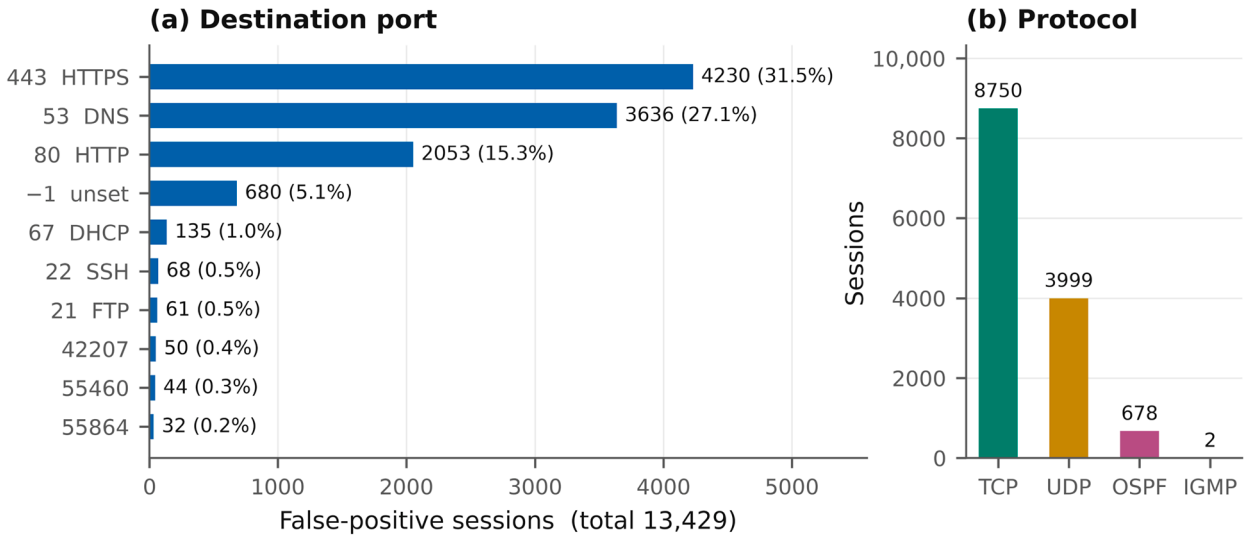


Figure 4. Destination port distribution of false-positive sessions in CICDDoS2019 (13,429 sessions in total). HTTPS, DNS, and HTTP account for 73.9% of the total.

SYN flooding is characterized by a high volume of incomplete connection attempts that stress the server's connection state [53,54]. In a SYN flood attack, the attacker transmits a large volume of SYN packets to exhaust the server's connection queue, and the server responds with RST packets for sessions whose connection setup failed. The SYN to RST pattern therefore correlates strongly with the typical termination behavior of attack sessions. To verify this, an additional comparative experiment with the SYN to FIN condition was conducted under identical parameters (pc = 4, strict matching disabled). At the session level, the two conditions produced a similar false-positive rate (SYN to RST: 0.48%, 13,429/2,771,186; SYN to FIN: 0.44%, 52,712/11,992,889), so session-level FPR alone does not distinguish the two flag conditions. The practical difference instead appears at the segmentation level: the SYN to FIN condition produced 4702 attack-interval files, 3.2 times more than the 1485 files produced by SYN to RST, because ordinary connection teardowns are far more likely to end in FIN than in RST. Of these files, 90.3% (1341/1485) of the SYN to RST files contained at least one true-positive attack session, versus only 72.8% (3423/4702) of the SYN to FIN files, and 27.2% (1279/4702) of SYN to FIN files contained no ground-truth attack session at all, compared with 9.7% (144/1485) for SYN to RST. This experiment shows that the SYN to RST combination's advantage over SYN to FIN lies not in session-level classification accuracy but in producing far fewer, more precisely bounded attack-interval files, a property that bears directly on the segmentation objective of Stage 2. This is reported as a dataset-specific empirical observation rather than a universal property of SYN floods.

A total of 1485 attack intervals were identified, achieving pair-level precision of 100% (2/2) and pair-level recall of 100% (2/2). A detailed analysis of system performance indicators such as processing time, memory, and CPU utilization is given in Section 5.5.

5.3. PortScan Attack Detection Experiment (CICIDS-2017 Friday)

This section describes the results of a PortScan attack detection experiment conducted on CICIDS-2017 Friday-WorkingHours.pcap (8.07 GiB in the classic format). The detection configuration applied Mode 4 (TCP flag) with the SYN to RST pattern, a minimum packet count (pc) threshold of 2, and strict matching enabled. In the experimental environment, the attacker IP was identified as 172.16.0.1 and the victim IP as 192.168.10.50. The total processing time was 5628 s (approximately 1.56 h), the average CPU utilization was 2.42%,

and the run completed within a 4 GB heap ceiling with a peak heap usage of 853 MB and a peak process RSS of 1502 MB.

PortScan detection was carried out in two stages: Stage 2 segmentation and post-processing classification of the feature CSV (Section 4.3). In Stage 2, applying strict matching in Mode 4 (Equation (7)) excluded sessions containing SYN + ACK (0 × 12) and extracted only pure SYN (0 × 02) sessions, which yielded 171,405 sessions in total. At the individual session level, however, PortScan sessions and normal Kerberos and LDAP sessions exhibit the same SYN to RST pattern, so the two types cannot be distinguished by the flag condition alone. Stage 2 therefore does not detect port scans: with the SYN to RST rule it only delimits candidate intervals, and the scan is identified by the cross-session rule of Algorithm 3. The basis for identifying PortScan lies not in the flag pattern of a single session but in the cross-session statistic that connection attempts to many destination ports are repeated from the same IP pair.

To resolve this, the feature-based post-processing classification of Section 4.3 was applied. The PortScan classification conditions of Table 5 are protocol = TCP (6), total packets per session (total_packet_count) of 5 or fewer, number of packets carrying a payload (payload_packet_count) equal to 0, and 50 or more unique destination ports for the same (source_ip, destination_ip) pair. Here, IP pairs are merged into canonical pairs by sort-order normalization, so scan traffic in the attacker-to-victim direction and reverse-direction response (RST) traffic are counted as a single pair. As a result, the only detected pair was 172.16.0.1 and 192.168.10.50, with zero false-positive pairs. Pair-level precision was measured at 100% (1/1) according to Equation (12), pair-level recall at 100% (1/1) according to Equation (13), and session-level precision at 100% (20,821/20,821). The attacker 172.16.0.1 scanned 999 unique ports on the victim, and 1049 when both directions of the pair are counted as in the rule, whereas the second-ranked pair under the same filter had 37.

Session-level recall was measured at 13.1% (20,821/158,930) according to Equation (13), but this stems from differences in the session definitions and counting schemes used by the proposed framework and existing tools. Because the calculated recall is low, operational use should be accompanied by a procedure that retrieves all traffic of the relevant IP pair on the basis of the pair-level detection result. In the proposed framework, a session is defined in units of the SYN to RST flag pattern, whereas in existing tools such as CICFlowMeter, a session is defined as a timeout-based flow, so the two units are not directly commensurable [55,56]. With the flow mapping of Section 5.1.3, the detected sessions cover 7.5% (11,953/158,930) of the labelled PortScan flows, and only 7.6% of these flows fall inside an attack interval at all, so the low recall arises in Stage 2, which delimits intervals only around SYN to RST triggers.

A threshold stability analysis was conducted to examine the influence of the classification thresholds of Table 5 on detection performance. Figure 5 shows the pair-level precision and recall measured while varying the number of unique destination ports for PortScan (unique_dst_ports, 10 to 100), the upper bound on total packets per session (total_packet_count, 1 to 20), and the session count for DoS (session_count, 1000 to 50,000). Pair-level precision and recall were 100% from 40 destination ports, a packet cap of 2 and 2000 sessions upward, so the operating points (50, 5 and 10,000) lie inside this range. Below these values, normal pairs were added (pair-level precision 4.2% at 10 ports and 50% at 1000 sessions), and a packet cap of 1 detected no pair. Session precision nevertheless stayed at or above 95.1% wherever a pair was detected, because the added pairs contribute few sessions compared with the attack pair (1069 of 21,890 at 10 ports). Because the thresholds were chosen and analyzed on the same data, this is a stability analysis and does not replace independent validation (Section 5.6.5).

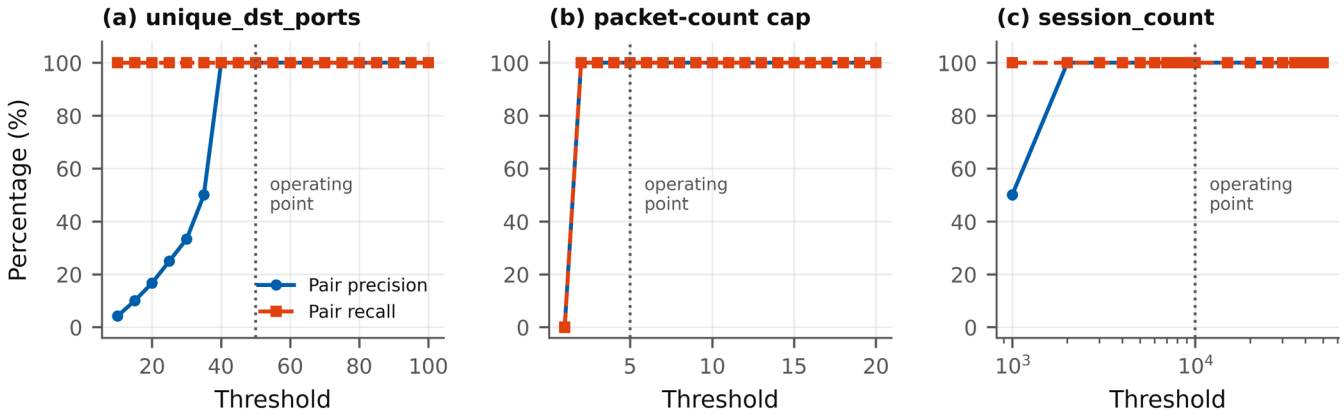


Figure 5. Stability analysis of the classification thresholds. (**a**) PortScan unique_dst_ports, (**b**) upper bound on PortScan total_packet_count, (**c**) DoS session_count. The gray dotted lines indicate the operating points used in this paper (50, 5, and 10,000). Pair-level precision and recall are 100% from 40 ports, a cap of 2 and 2000 sessions upward; session precision stays at or above 95.1% wherever a pair is detected.

5.4. DoS Attack Detection Experiment (CICIDS-2017 Wednesday)

This section describes the results of a DoS attack detection experiment conducted on CICIDS-2017 Wednesday-workingHours.pcap (12.28 GiB in the classic format). The dataset contains four types of DoS attack traffic: Hulk (231,073 instances), GoldenEye (10,293 instances), Slowloris (5796 instances), and SlowHTTPTest (5499 instances). The detection configuration applied Mode 4 (TCP flag), the SYN to RST pattern, a minimum packet count (pc) threshold of 4, and strict matching enabled. The total processing time was 5603 s (approximately 1.56 h), the average CPU utilization was 2.47%, and the run completed within a 4 GB heap ceiling with a peak heap usage of 906 MB and a peak process RSS of 1343 MB.

In Stage 2, filtering based on the SYN to RST pattern extracted 314,252 sessions in total. Next, the DoS classification conditions defined in Table 5 were applied in the feature CSV post-processing step (Section 4.3). The classification conditions judge a case to be an attack when the protocol is TCP (6), the destination port (destination_port) is one of {80, 443, 8080}, and the session count (session_count) for the same (source_ip, destination_ip, destination_port) triple is 10,000 or more. Applying these conditions yielded a single detected IP triple (172.16.0.1 to 192.168.10.50:80) with zero false positives. Pair-level precision of 100% (1/1), pair-level recall of 100% (1/1), and session-level precision of 100% (167,305/167,305) were achieved. The second-ranked pair by session count in normal traffic (normal HTTPS traffic) had 1002 sessions, a roughly 167-fold difference compared with the detected attack traffic (167,305 sessions), which confirms that the threshold of 10,000 clearly separates normal traffic from attack traffic.

Session-level recall was measured at 66.2% according to Equation (13). The number of attack sessions in the published ground truth for Wednesday is 252,672, which comprises the four DoS attacks listed above (252,661 flows) together with 11 Heartbleed flows, whereas the number of detected sessions is 167,305. This is because filtering on destination_port = 80 excluded both reverse-direction sessions from victim to attacker and sessions using ephemeral ports from the detection scope. As in Section 5.3, however, pair-level recall was 100%, so all traffic of the detected pair can be retrieved with the same procedure. With the flow mapping of Section 5.1.3, the detected sessions cover 91.1% (230,224/252,661) of the labelled DoS flows: 98.3% for Hulk, 28.6% for SlowHTTPTest, 18.9% for Slowloris and 4.6% for GoldenEye.

To show how the detected sessions are distributed among the four DoS types, each detected session was assigned to the published attack window of each tool, with a margin of two minutes. Hulk accounted for 164,827 sessions (71.3% of its 231,073 labeled flows), SlowHTTPTest for 1125 (20.5% of 5499), Slowloris for 1008 (17.4% of 5796) and GoldenEye for 345 (3.4% of 10,293); no detected session fell outside the four windows. All four tools used the same attacker, victim and port, so their sessions were counted in one triple, and the triple exceeded the session_count threshold because of Hulk alone. None of the other three types would have been detected on its own: Slowloris and SlowHTTPTest produced about 1000 sessions each and GoldenEye 345, an order of magnitude or more below the threshold of 10,000. Because the labels are timeout-based flows, these ratios are indicative only. The dependence on session frequency is a limitation of the current rule (Section 5.6.5).

5.5. System Performance Evaluation

This section analyzes the system performance measurements for the three experimental datasets with reference to Table 6. Performance is evaluated in terms of three metrics: processing time, memory usage, and CPU utilization.

5.5.1. Processing Time

Processing time depends on the size of the input, on whether it is supplied in chunks and, for the remainder-file variant of Algorithm 1, on the number of attack intervals. All inputs and outputs were on the internal disk of the platform of Section 5.1.1.

The CICDDoS2019 dataset is distributed as 965 chunk files (about 190 MiB each). With the remainder-file variant, end-to-end processing of 179.44 GiB took 1402 s (approximately 23.4 min), an average throughput of about 131.0 MiB/s (7.8 s/GiB). The 1485 attack intervals lie in 221 of the 965 files, and the remainder files written for them amount to 117.4 GiB, 0.65 times the input. The single-pass variant produced the same 1485 byte ranges and 2,771,186 Stage 3 sessions and took 706 s (260.1 MiB/s, 3.9 s/GiB).

CICIDS-2017 Friday (8.07 GiB) and Wednesday (12.28 GiB) are each provided as a single file. With the remainder-file variant, every detected interval causes the rest of the file to be copied and scanned again (Equation (4)). Friday produced 2157 attack-interval files and Wednesday 1760, and the variant wrote 6019 GiB and 5856 GiB in total, 746 and 477 times the input size, with a peak of 16.1 and 24.5 GiB of temporary disk space at the second split. End-to-end processing took 5628 and 5603 s (697 and 456 s/GiB). The single-pass variant reads each capture once and writes only the attack-interval files (1.91 and 2.54 GiB). It produced the same 2157 and 1760 byte ranges and the same 171,405 and 314,252 Stage 3 sessions, and took 363 and 369 s end to end (45 and 30 s/GiB), of which segmentation took 50.5 and 78.4 s and Stage 3 the rest. The Detected Attacks item of Table 6 denotes the number of attack-interval files produced by segmentation, an indicator of segmentation workload that is distinct from the pair-level detection result.

5.5.2. Memory Usage

In terms of memory usage, the byte-offset zero-copy design of the framework was empirically validated. Equation (2) states theoretically that $M_{split} = O(1)$, meaning that the memory complexity of the segmentation stage is independent of the input file size $|F|$, and this was confirmed quantitatively in the present experiments. The change in memory before and after processing, used as an indicator in earlier measurements, is not reported here: garbage collection returns the heap to its baseline by the end of a run, so the change does not reflect the peak during processing. This paper therefore uses the minimum heap that allows the run to complete, rather than the before-and-after change, as the indicator of end-to-end memory behavior. Memory was measured by sampling

heap usage (MemoryMXBean), non-heap usage, direct buffers (BufferPoolMXBean), and GC time at one-second intervals, while the process RSS was collected as the working set at the same interval from a separate process. Direct buffer usage was 0 MB throughout, confirming that jNetPcap's native allocation does not use off-heap direct memory. For CICDDoS2019 under a 4 GB heap ceiling, the average actual heap usage was 232 MB and the maximum was 1225 MB. This is the effect of the zero-copy structure, which references the relevant interval directly using only byte-offset pointers instead of loading the entire file into memory.

To quantify the memory requirement, the heap ceiling (-Xmx) was lowered step by step and the minimum heap that allowed a run to complete was measured. CICDDoS2019 (965 chunks, 179 GiB) completed at every ceiling tried, down to 1 GB. At a 4 GB ceiling the maximum actual heap usage was 1225 MB, the maximum process RSS was 1726 MB, and the GC overhead was 1.04% of the total processing time. Raising the ceiling to 32 GB expands the committed heap to 4688 MB, while the processing time stays within 3.3% across the four ceilings (1358 to 1402 s). The committed heap size therefore follows the heap ceiling, and limiting the ceiling to 1 GB does not slow processing. Because the minimum heap is determined by run completion, it is independent of the sampling interval. Figure 6 shows the peak heap usage and peak process RSS of each dataset at a 4 GB ceiling, together with the lowest ceiling at which it completed (Table 6). Figure 7 plots memory and processing time against the heap ceiling, and Figure 8 shows the memory time series of the 4 GB run at one-second resolution.

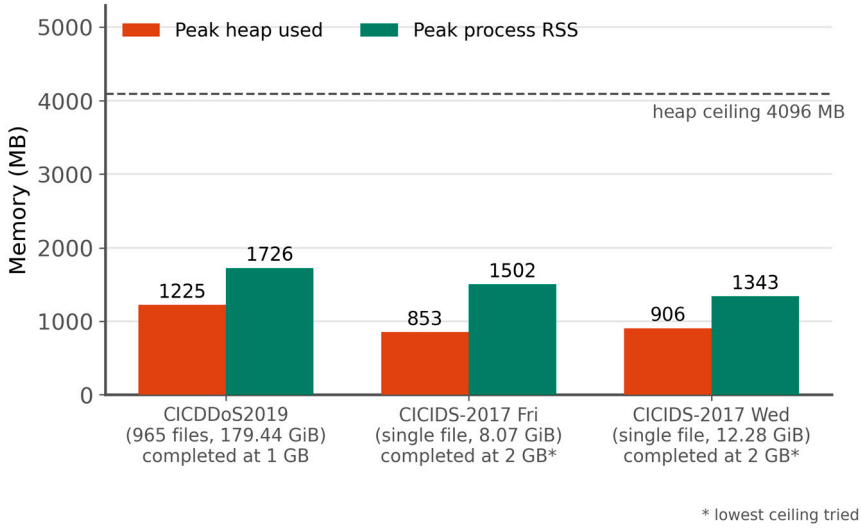


Figure 6. Peak heap usage and peak process RSS of the three datasets at a 4 GB heap ceiling, and the lowest ceiling at which each run completed. The single files do not need more memory than the 179 GiB chunked input.

The segmentation module alone uses only about 30 MB of heap for an 8 GiB input, so the O(1) property of Equation (2) is preserved, and in end-to-end processing the session state of the detection stage and of Stage 3 is added to this constant cost. In the CICDDoS2019 run with a 4 GB ceiling, the average process RSS was 761 MB. With the same instrumentation sampling at one-second intervals, the CICIDS-2017 Friday and Wednesday runs showed peak heap usage of 853 MB and 906 MB, average heap usage of 238 MB and 307 MB, average process RSS of 758 MB and 976 MB, and garbage-collection overhead of 0.017% and 0.022% of total processing time. The peaks for the two single files are of the same order as for the chunked CICDDoS2019 input (1225 MB): Stage 3 receives only the attack-interval files,

whose size is bounded by the attack interval rather than by the input file, and the detection stage keeps only the sessions of the range currently being scanned (Equation (3)).

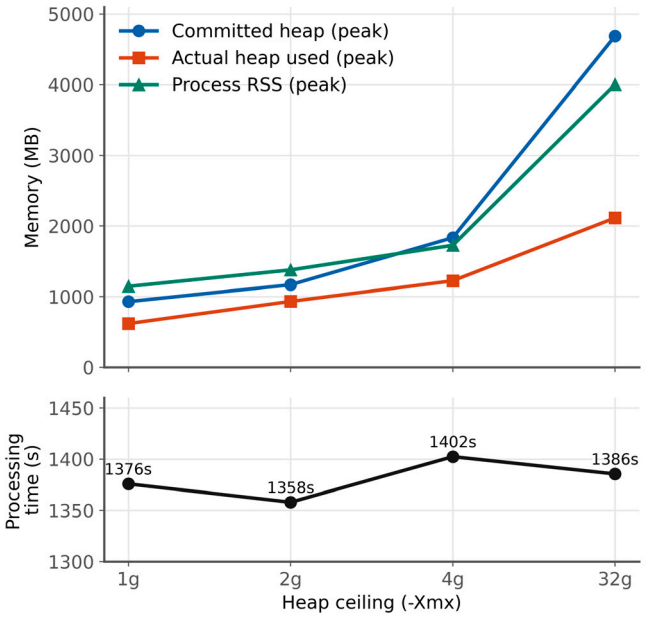


Figure 7. Memory and processing time as a function of the heap ceiling (CICDDoS2019). Raising the ceiling mainly increases the committed heap, while the processing time stays within 3.3%. The input was on the internal disk. The run was completed at all four ceilings.

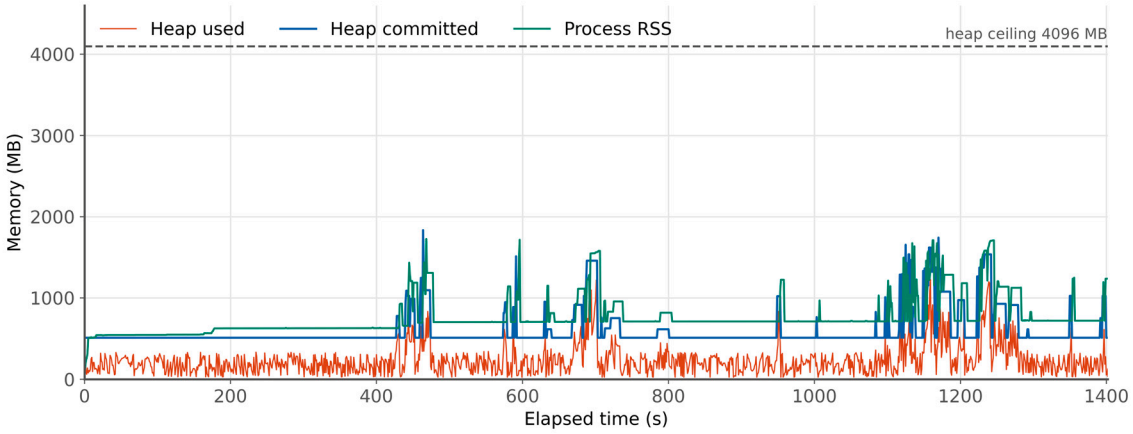


Figure 8. Memory time series during processing of CICDDoS2019 (179 GiB, 965 chunks) with -Xmx 4 GB. Actual heap usage, committed heap, and process RSS were measured at one-second intervals. Usage rises at each chunk boundary and then returns to the baseline, and the entire run stays below the 4096 MB ceiling.

For a single large capture, the single-pass variant removes the dependence of processing time on the number of attack intervals. The heap needed for an unsplit file can be estimated from the number of unique flows in one continuous scan range, following Equation (3); splitting a capture into pieces of about 1 GB (for example, with editcap or tcpdump -C) bounds this range. Splitting has a cost: a session that crosses a piece boundary is recorded as two sessions, and an attack interval that crosses a boundary is divided between two files. Chunked input also allows several pieces to be processed by separate instances in parallel, which was not evaluated in this paper.

5.5.3. CPU Utilization

CPU utilization was measured as the process CPU load of the framework, normalized over all 32 logical cores of the experimental platform. The averages were 3.68% for CICDDoS2019, 2.42% for CICIDS-2017 Friday, and 2.47% for CICIDS-2017 Wednesday, corresponding to roughly 0.8 to 1.2 fully utilized cores and therefore consistent with the single-threaded design of the analysis logic. The observed maxima of 12.35%, 8.82%, and 7.85% correspond to approximately 4.0, 2.8, and 2.5 cores and thus exceed what a strictly single-threaded process could generate on its own; these peaks are attributable to the background feature extraction thread, which runs alongside Stage 2 (Section 3.4), and to the parallel G1 garbage collector and file input and output, rather than to parallelism within the analysis logic of either stage. Sustained utilization nevertheless remained on the order of a single core throughout processing, which means that the system can run concurrently with other analysis jobs while sharing resources when used as a background analysis tool in a network forensic environment. Figure 9 compares the end-to-end throughput of the three datasets.

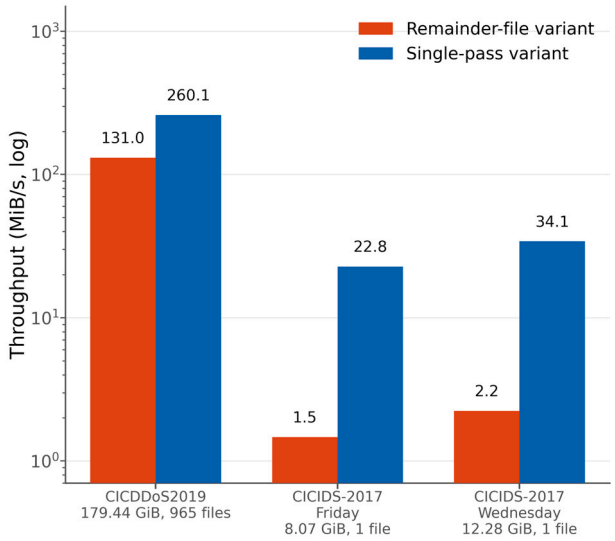


Figure 9. End-to-end performance comparison. Throughput of the remainder-file and single-pass variants of Algorithm 1, all with the input on the internal disk. CICDDoS2019 consists of 965 chunks of about 190 MiB; the CICIDS-2017 captures are single files.

With the remainder-file variant, CICDDoS2019 (179 GiB in 965 chunk files of about 190 MiB) was processed at 131.0 MiB/s and the single CICIDS-2017 files at 1.5 and 2.2 MiB/s. With the single-pass variant, the rates were 260.1, 22.8 and 34.1 MiB/s (Figure 9).

5.5.4. Design Rationale for the Single-Threaded Processing Structure

The framework deliberately adopts a single-threaded sequential processing structure even in a multi-core environment. This stems from three design requirements. First, attack-triggered segmentation and session reconstruction depend strongly on the chronological order of packets. Flag transitions from SYN to RST and the start and end of an attack interval can be identified accurately only when packets are processed in their arrival order, and the nondeterministic execution order that arises under parallel processing can compromise the integrity of such state transition tracking. Second, session key normalization and cumulative statistics computation are inherently stateful operations, so lock contention and synchronization costs arise when multiple threads access the same session state concurrently. Third, the bottleneck of the segmentation stage of this system lies in disk input and output (I/O bound) rather than in computation. Byte-offset-based zero-copy segmentation

minimizes CPU computation and relies on sequential file reads, so the overhead incurred by context switching and synchronization outweighs the throughput gain obtained from thread-level parallelization.

Indeed, the fact that the average CPU utilization stayed in the 2 to 4% range (2.42 to 3.68% of a 32-logical-core platform, that is, on the order of a single fully utilized core) in all three experiments in Section 5.5 supports the conclusion that the system does not saturate the available computational resources. The short peaks of up to 12.35% come from the concurrent feature extraction thread, the parallel garbage collector and input and output threads; the analysis logic of each stage remains single-threaded. The single-threaded structure is therefore not an architectural limitation but an intentional design decision that takes into account both processing correctness and the I/O-bound nature of the segmentation stage.

5.6. Discussion

5.6.1. Summary of the Functional Validation Results

As shown in Table 7, the framework proposed in this study was validated for the three attack types of DDoS, PortScan, and DoS using two public datasets (CICDDoS2019 and CICIDS-2017). Each experiment contains only one or two attack pairs, so the pair-level values of Table 7 (1/1 or 2/2) show that the attack pairs were found and are not evidence of accuracy. Session-level precision was 99.52% for DDoS (direct detection in Stage 2) and 100% for PortScan and DoS over the classified sessions, and 21.05% and 75.88% over all Stage 3 sessions (note to Table 7). Session-level recall for DDoS was excluded from the computation because the ground truth of CICDDoS2019 is provided in units of CICFlowMeter-based timeout flows, so its session counting scheme differs fundamentally from that of the proposed framework, which defines a session in units of the SYN to RST flag transition pattern. Instead, Table 7 reports that both attack pairs were found (2/2). In particular, the fact that all three attack types were handled using the same Mode 4 (TCP flag) detection logic by changing only the detection parameters (the pc threshold, the strict setting, and the feature-based post-processing classification conditions) shows that one mode covered the three evaluated attack types with different parameters. This is, however, a functional confirmation on three experiments with two datasets and does not imply generalization to unseen attack types or to independently collected corpora. In addition, a file-level analysis of the 1485 attack-interval files produced in the DDoS experiment showed that 90.3% (1341/1485) of the files contained attack sessions. The 144 files without attack sessions consist mostly of normal background traffic from periods in which no attack occurred and were extracted together at segmentation boundaries. The corresponding ratios for the other two datasets are 70.6% (1243/1760) for CICIDS-2017 Wednesday and 3.5% (76/2157) for CICIDS-2017 Friday. The low ratio for Friday is a direct consequence of the attack type rather than of a segmentation failure: a PortScan session consists of a single short SYN-to-RST exchange carrying no payload, so the great majority of the intervals that Stage 2 delimits around such a trigger contain only the surrounding background traffic. This is also why PortScan cannot be identified from a single session and requires the cross-session statistics of Section 4.3; the file-level containment ratio should therefore be read as a property of the attack profile, not as a measure of segmentation precision. The Friday attack pair was still found.

The complementary structure of Stage 2 segmentation and feature-based post-processing classification was also confirmed in these experiments. Stage 2 is optimized for large-scale segmentation but has low discriminative power at the individual session level, and post-processing classification compensates for this with cross-session statistics, so that together they constitute an end-to-end analysis procedure.

5.6.2. Comparison with Existing Tools

A comparison with the existing network analysis tools presented in Table 2 confirms the distinctiveness of the proposed framework. CICFlowMeter provides more than 80 flow-statistical features [18,19,57] but offers neither PCAP segmentation nor integrated attack detection. As described in Section 1.1, Wireshark, SplitCap and Zeek [5–8] each cover part of this workflow, but none of them natively produces attack-interval PCAP files together with the fixed 49-feature CSV schema used in this study. In contrast, the proposed framework provides, within a single system, an end-to-end automated pipeline spanning attack detection, byte-offset segmentation, extraction of 49 multi-protocol features, and CSV generation. It maintains the O(1) memory complexity of Equation (2) in the segmentation stage, and for input supplied in chunks, the end-to-end processing is also carried out within the heap ceiling. For a single large file, the session-tracking state of the detection stage covers one continuous scan range (Equation (3)); in the two single-file experiments, the peak heap stayed below 1 GB, but captures with many more concurrent flows may need a larger heap or prior chunk-wise partitioning.

The processing cost of the feature extraction stage was measured directly against CICFlowMeter v4.0 [58] and Zeek [7,8] on the full CICIDS-2017 Friday (8.07 GiB) and Wednesday (12.28 GiB) files, on the platform of Section 5.1.1 (Table 8). CICFlowMeter and Stage 3 of the proposed system were run as stand-alone Java programs with the same JDK and JVM options (-Xms512m -Xmx32g -XX:TieredStopAtLevel=1). Zeek was run with its default scripts in WSL2 on the same machine, with the input copied to the virtual machine's local file system beforehand. Process CPU and resident set size were sampled every 1 to 1.5 s, and CPU utilization is process CPU normalized over the 32 logical cores, as in Table 6. Zeek was the fastest and lightest tool (8.0 to 9.0 s/GiB, peak RSS 200 to 401 MB). Stage 3 took 0.68 and 0.78 times as long as CICFlowMeter (Friday and Wednesday, respectively), but its peak RSS was about seven times higher (25,349 and 29,307 MB against 3547 and 4188 MB). This is because the whole capture was passed to Stage 3 as one input: by default, Stage 3 applies no idle timeout (Section 4.2.1), so every session is kept until the end of the file. With the optional FIN/RST termination and a 120 s idle timeout, peak RSS fell to 15,537 and 18,041 MB at about the same processing time (157.7 and 238.5 s), because records are summarized and released when they close. In the pipeline, Stage 3 receives only attack-interval files, and the end-to-end peak RSS was 1502 and 1343 MB (Table 6). With the remainder-file variant, the end-to-end rates (697 and 456 s/GiB) are dominated by repeated scanning in Stage 2 (Section 5.5.1); the single-pass variant processes the two files end to end at 45 and 30 s/GiB, including segmentation, which CICFlowMeter and Zeek do not perform. Stage 3 also used more than one core (average 9.76% and 10.73%, about 3.1 and 3.4 cores; maximum 32.80% and 42.16%), although its parsing logic runs in a single thread. With all sessions kept in a heap of up to 32 GB, the parallel G1 garbage collector (Section 5.5.3) and the JIT compiler threads of the JVM run alongside that thread; their CPU time was not measured separately.

This comparison requires careful interpretation, however, because the design goals of the stages differ. The proposed framework is divided into Stage 2 (attack-interval segmentation) and Stage 3 (feature extraction), and the two stages have different optimization objectives. Stage 2 prioritizes processing speed (50.5 and 78.4 s for the two files in the single-pass variant), whereas Stage 3 is designed to prioritize the accuracy and multidimensionality of the extracted information over processing speed. Because CICFlowMeter and Zeek do not segment the input, they are comparable only with Stage 3 of the proposed system. In that comparison, the higher memory usage of Stage 3 results from retaining all sessions without an idle timeout, and its additional output consists of the TLS, DNS and HTTP fields and the payload statistics summarized in Table 9.

Table 8. Full-file benchmark of feature extraction on the CICIDS-2017 files on identical hardware.

Tool	Input	Time (s)	s/GiB	CPU Avg/Max (%)	Peak RSS (MB)
CICFlowMeter v4.0	Friday (8.07 GiB)	221.6	27.5	3.45/7.51	3547
CICFlowMeter v4.0	Wednesday (12.28 GiB)	278.0	22.6	3.27/8.04	4188
Zeek 8.0.10	Friday (8.07 GiB)	73.0	9.0	3.40/3.57	200
Zeek 8.0.10	Wednesday (12.28 GiB)	98.3	8.0	3.37/3.54	401
Proposed, Stage 3 only	Friday (8.07 GiB)	149.7	18.6	9.76/32.80	25,349
Proposed, Stage 3 only	Wednesday (12.28 GiB)	217.3	17.7	10.73/42.16	29,307
Proposed, Stage 3 with FIN/RST and 120 s idle timeout	Friday (8.07 GiB)	157.7	19.5	10.83/28.38	15,537
Proposed, Stage 3 with FIN/RST and 120 s idle timeout	Wednesday (12.28 GiB)	238.5	19.4	11.47/32.31	18,041

Note: Inputs are the classic-format PCAP files of Section 5.1.2. CICFlowMeter v4.0 (commit 98a5eba) and Stage 3 ran on JDK 17 with -Xms512m -Xmx32g -XX:TieredStopAtLevel=1; Zeek 8.0.10 ran in WSL2 with its default scripts and the input on the virtual machine's ext4 file system. CPU is process CPU normalized over 32 logical cores. The outputs differ: Zeek writes protocol logs (conn.log and others), CICFlowMeter writes 84 flow features per flow, and Stage 3 writes the 49 session features of Table 4. End-to-end values of the proposed pipeline are given in Table 6. The last two rows were measured on a later day together with a repeat of the other Java rows (CICFlowMeter 205.9 and 270.8 s; Stage 3 153.9 and 233.6 s), within 8% of the values shown. With the timeout option, Stage 3 produced 582,096 and 721,250 session records, against 396,968 and 226,725 without it and 695,257 and 777,037 CICFlowMeter flows.

Table 9. Comparison of functional differences between CICFlowMeter v4.0 and the proposed system.

Comparison Item	CICFlowMeter v4.0	Proposed System
Number of output features	84	49
Nature of features	Flow statistics only	Flow + application layer + payload
Flow/time/TCP flag statistics	✓	✓
TLS handshake (cipher, certificate, SAN)	—	✓
DNS query (suffix, TTL, character distribution)	—	✓
HTTP metadata (UA, content-type, response code)	—	✓
Payload entropy/character statistics	—	✓
Integrated attack detection	—	✓
Attack-interval segmentation (Stage 2)	—	✓
Single integrated workflow	—	✓

Note: ✓, supported; —, not supported.

5.6.3. Detection and Splitting Baselines

Two baselines suggested for the evaluation were run on the same files and hardware (Table 10). For detection, Zeek 8.0.10 was run with a port-scan policy. Zeek 8 no longer ships the scan.zeek policy of earlier releases, so the logic of that policy was loaded as a script: a rejected, reset or unanswered connection attempt is counted per (scanner, victim) and per (scanner, port), and a Port_Scan notice is raised at 15 unique ports on one host within 5 min (an Address_Scan notice at 25 hosts on one port). On Friday, Zeek raised one Port_Scan notice, for 172.16.0.1 scanning 192.168.10.50, and four Address_Scan notices for internal hosts that contacted at least 25 hosts on port 443. On Wednesday it raised no notice for the DoS pair, because Zeek has no policy for this kind of flooding, and four Address_Scan notices of the same kind. Zeek thus found the scanning pair, as the cross-session rule of Algorithm 3 does, in 76 s for the full file, but it writes neither attack-interval PCAP files nor session features.

Table 10. Detection and splitting baselines on the CICIDS-2017 files (same hardware and disk).

Task	Tool	Friday (8.07 GiB)	Wednesday (12.28 GiB)
Scan detection	Zeek 8.0.10, scan.zeek logic	76.2 s; Port_Scan notice for the attack pair; 4 other notices	101.3 s; no notice for the DoS pair; 4 other notices
Scan and DoS detection	Proposed, Algorithm 3 on Stage 3 output	attack pair only (Table 7)	attack triple only (Table 7)
Interval cutting	editcap 4.6.0, one call per interval	1.5 s per call (mean of 20); about 0.9 h for 2157 intervals	2.1 s per call (mean of 20); about 1.1 h for 1760 intervals
Pair extraction	tshark 4.6.0, display filter	77.8 s; one file (1.24 GB)	106.3 s; one file (2.19 GB)
Detection and interval cutting	Proposed, single-pass Stage 2	50.5 s; 2157 files	78.4 s; 1760 files

Note: Zeek ran in WSL2 with its default scripts and the scan logic, with the input on the virtual machine's file system; the time includes all default logs. editcap outputs were byte-identical with the attack-interval files of the framework for the 20 sampled intervals per file; the hour values are estimates from a linear fit of call time against the last packet number of the interval. tshark and editcap ran on Windows from the internal disk.

For splitting, the attack intervals found by Stage 2 were cut from the original captures by packet range with editcap 4.6.0. For 20 intervals spread over each file, the output was byte-identical to the attack-interval file of the framework in every case. Each call writes one file and reads the capture up to the end of the selected range, so a call took 0.03 to 2.4 s on Friday and 0.03 to 3.2 s on Wednesday depending on the position of the interval. A linear fit of these times over all intervals gives about 0.9 and 1.1 h to cut all 2157 and 1760 intervals, against 50.5 and 78.4 s for single-pass segmentation including detection; editcap also needs the packet ranges, which come from a detection step. A tshark 4.6.0 display filter extracted all packets between 172.16.0.1 and 192.168.10.50 into one file in 78 and 106 s (1.24 and 2.19 GB); this requires the attacker and victim to be known and does not separate attack intervals.

5.6.4. Classification Experiment

To test whether the extracted features support supervised classification, XGBoost 3.2.0 [9] was trained on the Stage 3 sessions of the Friday and Wednesday single-file runs. Each session received the CICIDS-2017 label of the flows mapped to it (Section 5.1.3): attack if any mapped flow carries an attack label and benign if all mapped flows are benign; sessions without a mapped flow (38,513 and 110,959) were excluded. The task is binary, with the 79 inputs of Section 4.2.5. Five-fold cross-validation was grouped by attack-interval file, so that the sessions of one file never appear in both the training and the test folds. The model used 300 trees of depth 6 with a learning rate of 0.1 and row and column subsampling of 0.8. Table 11 reports the mean and standard deviation over the folds, together with the cross-session rules of Algorithm 3 scored against the same labels.

The classifier reached an F1 score of 0.980 on Friday and 0.990 on Wednesday, against 0.70 and 0.93 for the rules, whose precision was higher (0.993 and 0.998) and recall lower (0.60 and 0.87). The experiment has two limitations. The sessions come from the attack-interval files selected by Stage 2, so the class balance differs from that of the full captures. The most important input on Wednesday was the presence of an HTTP content type (share of total gain 0.66), which mainly separates the Hulk requests from the rest of the traffic and may not transfer to other captures. The experiment shows that the session features can be used directly as model input; it is not a comparison of detection methods.

Table 11. Binary classification of Stage 3 sessions with XGBoost (five folds grouped by attack-interval file).

Dataset	Sessions (Attack/Benign)	Model	Precision	Recall	F1	PR-AUC
Friday	132,877 (32,863/100,014)	XGBoost	0.968 ± 0.010	0.993 ± 0.004	0.980 ± 0.007	0.991 ± 0.005
Friday	132,877 (32,863/100,014)	Rule (Algorithm 3)	0.993 ± 0.003	0.60 ± 0.27	0.70 ± 0.27	—
Wednesday	203,293 (136,463/66,830)	XGBoost	0.990 ± 0.001	0.990 ± 0.014	0.990 ± 0.007	0.997 ± <0.001
Wednesday	203,293 (136,463/66,830)	Rule (Algorithm 3)	0.998 ± <0.001	0.87 ± 0.01	0.93 ± 0.01	—

Note: Labels are the CICIDS-2017 flow labels mapped to sessions (Section 5.1.3); Friday attack sessions are PortScan (11,990), DDoS (20,672) and Bot (201), Wednesday attack sessions are the four DoS types and Heartbleed. The rule row scores the sessions classified by Algorithm 3 in each test fold; on Friday the rule targets PortScan only, so DDoS and Bot sessions count as missed. Values are mean ± standard deviation over five folds.

5.6.5. Limitations

The platform is rule-based. Each detection mode requires the analyst to choose parameters such as flag masks, ports, payload patterns and thresholds, and the evaluation covers three attack types (DDoS, PortScan and DoS) on two public datasets. The results do not show that the same settings, or the parameter-based design in general, carry over to other attack types or to independently collected traffic.

The classification thresholds $\theta p = 50$ and $\theta s = 10{,}000$ are operating points chosen on the evaluated data. The analysis in Figure 5 uses the same data and therefore shows stability around these points; it is not an independent validation. To check transfer to another corpus, the Wednesday configuration (Mode 4, SYN to RST, $pc = 4$, strict matching, $\theta s = 10{,}000$) was applied unchanged to the captures of the victim web server in CSE-CIC-IDS2018 [52]. On 15 February 2018 (GoldenEye and Slowloris), Stage 2 produced no attack interval, so neither attack was detected. On 16 February 2018 (SlowHTTPTest and Hulk), Stage 2 likewise produced no attack interval. The settings chosen on CICIDS-2017 therefore do not carry over to another corpus without retuning.

The DoS rule counts sessions per (source, destination, port) triple. Low-rate attacks such as Slowloris and SlowHTTPTest keep a few long-lived connections open. In the Wednesday capture, they produced 1008 and 1125 detected sessions (Section 5.4), an order of magnitude below θs, and were counted as detected only because they shared the attacker, victim and port with Hulk. GoldenEye also produced only 345 detected sessions, so the detection of the triple rested on Hulk alone. The platform already extracts duration and inter-arrival features (flow_start_time, flow_end_time and flow_iat_list), which could support a duration-based rule; this is left as future work.

The current implementation parses IPv4 only; packets without an IPv4 header are skipped, so IPv6 traffic is not analyzed. Both evaluation corpora contain IPv4 traffic only. Byte offsets are accumulated from the per-record capture length, so segmentation boundaries do not depend on link-layer encapsulation. In a test in which a Wednesday segment (7901 packets) was re-encoded with an 802.1Q tag on every frame, Stage 2 produced the same attack interval and Stage 3 the same 524 sessions. Payload-derived features, however, differed: the payload length is derived assuming an untagged Ethernet header, so the 4-byte tag was counted as payload, and TLS handshake fields were not recovered for 13 of 103 TLS sessions. VLAN-tagged captures should therefore have their tags removed before feature extraction (for example, with tcprewrite –enet-vlan=del). The session key does not include the VLAN identifier, so identical 5-tuples on different VLANs are merged, and hosts behind NAT are not distinguished. In Stage 2, Mode 4 resets the end-flag counter whenever the start flag matches again (Equation (8)), so reused 5-tuples are counted per connection

attempt; by default, Stage 3 applies no idle timeout, so packets sharing a 5-tuple within one segmented file form one session record (Section 4.2.1).

Both variants of Algorithm 1 clear the detection state at the end of each attack interval. When attacks by different IP pairs overlap in time, the start of the second attack can fall inside the interval cut for the first attack, so its interval may be split or incomplete.

6. Conclusions

This study presented, as a system contribution, the design and processing cost of a network traffic analysis system that integrates attack-triggered PCAP segmentation and multi-protocol session feature extraction into a single pipeline. The two core stages of the proposed system are attack-triggered segmentation and session-level feature extraction (Stages 2 and 3 of the pipeline in Section 3.2). Of its five detection modes, Mode 4 was evaluated and covered the three evaluated attack types through parameter settings; Modes 1, 2, 3 and 5 are implemented but not evaluated. The system also provides byte-offset-based zero-copy segmentation; extraction of 49 session-level features through methods including deep parsing of TLS, DNS, and HTTP; and full automation from file upload to CSV dataset generation.

In experiments with Mode 4 on CICDDoS2019 and CICIDS-2017, session-level precision was 99.52% for SYN flood traffic. For PortScan and DoS, the cross-session rules classified 20,821 and 167,305 sessions without false positives; over all Stage 3 sessions, precision was 21.05% and 75.88%, session recall was 13.1% and 66.2%, and flow-level recall against the published labels was 7.5% and 91.1%. The DoS settings did not detect the attacks in CSE-CIC-IDS2018 without retuning. In terms of system performance, the 179 GiB chunked dataset and the single 8.07 and 12.28 GiB captures were processed with a 2 GB heap ceiling (CICDDoS2019 also with 1 GB), with peak heap usage between 0.85 and 1.23 GB at a 4 GB ceiling. The single-pass variant of the segmentation produced the same attack-interval files as the remainder-file loop and reduced the end-to-end time for the two single captures from about 1.56 h to about 6 min and for the chunked CICDDoS2019 input from 1402 to 706 s. On the extracted features, an XGBoost classifier reached F1 scores of 0.98 and 0.99 with cross-validation grouped by attack-interval file.

This study adopted an analysis platform based on stored PCAP files to prioritize analytical precision and reproducibility. Because the proposed byte-offset segmentation and session feature extraction structure can be applied identically even if the input stage is replaced by live capture, future work will prioritize extension to a real-time streaming pipeline that proceeds from packet capture through segmentation and feature extraction to classification. We also plan to supplement the detection of low-rate DoS attacks (Slowloris, SlowHTTPTest, and others) through session-duration-based detection rules, to re-examine the Stage 2 settings and thresholds on independently collected traffic such as CSE-CIC-IDS2018; to validate additional attack types such as brute force, botnet, and web attacks; to evaluate Modes 1, 2, 3 and 5; to make the single-pass variant the default of the web interface; and to extend the classification experiment of Section 5.6.4 to complete captures and other corpora. The proposed framework is expected to serve as a practical foundational tool for network forensic analysis and for building training datasets for machine learning-based intrusion detection systems.

Author Contributions: Conceptualization, E.C. and M.C.; methodology, E.C.; software, E.C.; validation, E.C., S.H., D.T.A. and M.C.; formal analysis, E.C.; investigation, E.C.; resources, M.C.; data curation, E.C.; writing—original draft preparation, E.C.; writing—review and editing, E.C., J.H. and M.C.; visualization, E.C. and T.Q.D.; supervision, M.C.; project administration, M.C.; funding acquisition, M.C. All authors have read and agreed to the published version of the manuscript.

Funding: This research was jointly supported by the academic research program of Chungbuk National in 2025, jointly supported by the Regional Innovation System & Education (RISE) program through the Chungbuk Regional Innovation System & Education Center, funded by Chungcheongbuk-do and the Korea government (MOE), grant number 2026-RISE-11-014-02, and jointly supported by the Innovative Human Resource Development for Local Intellectualization program through the Institute of Information & Communications Technology Planning & Evaluation (IITP) grant funded by the Korea government (MSIT), grant number IITP-2026-RS-2020-II201462. During the preparation of this manuscript, the authors used generative artificial intelligence tools for English translation, language refinement, proofreading, and assistance in checking textual and reference consistency. All outputs were independently reviewed and verified by the authors. The authors take full responsibility for the accuracy and integrity of the final manuscript.

Data Availability Statement: The CICDDoS2019, CICIDS-2017 and CSE-CIC-IDS2018 datasets analyzed in this study are publicly available from the Canadian Institute for Cybersecurity [20,51,52]. The source code and measurement scripts of the platform were developed in a research project with confidentiality requirements and are not publicly available. The detection rules, parameters and measurement settings needed to repeat the experiments are given in Sections 3–5, and the results are reported in Tables 6–11.

Acknowledgments: This research was jointly supported by the academic research program of Chungbuk National in 2025, jointly supported by the Regional Innovation System & Education (RISE) program through the Chungbuk Regional Innovation System & Education Center, funded by Chungcheongbuk-do and the Korea government (MOE), grant number 2026-RISE-11-014-02, and jointly supported by the Innovative Human Resource Development for Local Intellectualization program through the Institute of Information & Communications Technology Planning & Evaluation (IITP) grant funded by the Korea government (MSIT), grant number IITP-2026-RS-2020-II201462. During the preparation of this manuscript, the authors used generative artificial intelligence tools for English translation, language refinement, proofreading, and assistance in checking textual and reference consistency. All outputs were independently reviewed and verified by the authors. The authors take full responsibility for the accuracy and integrity of the final manuscript.

Conflicts of Interest: The authors declare no conflicts of interest.

References

1. Cloudflare. *Cloudflare DDoS Threat Report for 2025 Q4*; Cloudflare Radar: San Francisco, CA, USA, 2026. Available online: https://radar.cloudflare.com/reports/ddos-2025-q4 (accessed on 24 August 2026).
2. Ministry of Science and ICT; Korea Internet & Security Agency (KISA). *Cyber Threat Trends Report for the First Half of 2025*; KISA: Naju, Republic of Korea, 2025. Available online: https://www.kisa.or.kr/20205/form?lang_type=KO&postSeq=1028 (accessed on 24 August 2026). (In Korean)
3. Sikos, L.F. Packet Analysis for Network Forensics: A Comprehensive Survey. *Forensic Sci. Int. Digit. Investig.* **2020**, *32*, 200892. [CrossRef]
4. The Tcpdump Group. Tcpdump and Libpcap. Available online: https://www.tcpdump.org (accessed on 24 August 2026).
5. NETRESEC AB. SplitCap: A Fast PCAP File Splitter. Available online: https://www.netresec.com/?page=SplitCap (accessed on 24 August 2026).
6. Wireshark Foundation. Wireshark: Network Protocol Analyzer. Available online: https://www.wireshark.org (accessed on 24 August 2026).
7. Paxson, V. Bro: A System for Detecting Network Intruders in Real-Time. *Comput. Netw.* **1999**, *31*, 2435–2463. [CrossRef]
8. Zeek Project. Invoking Zeek—Book of Zeek. Available online: https://docs.zeek.org/en/current/tutorial/invoking-zeek.html (accessed on 25 August 2026).
9. Chen, T.; Guestrin, C. XGBoost: A Scalable Tree Boosting System. In Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, San Francisco, CA, USA, 13–17 August 2016; pp. 785–794. [CrossRef]
10. Sly Technologies. jNetPcap: A libpcap Java Binding, Version 1.4. Available online: https://sourceforge.net/projects/jnetpcap/files/jnetpcap/1.4/ (accessed on 24 August 2026).
11. Oracle. FileChannel (Java SE 17 & JDK 17): Transfer to Method. Available online: https://docs.oracle.com/en/java/javase/17/docs/api/java.base/java/nio/channels/FileChannel.html (accessed on 25 August 2026).

12. Maier, G.; Sommer, R.; Dreger, H.; Feldmann, A.; Paxson, V.; Schneider, F. Enriching Network Security Analysis with Time Travel. *ACM SIGCOMM Comput. Commun. Rev.* **2008**, *38*, 183–194. [CrossRef]
13. Fusco, F.; Dimitropoulos, X.; Vlachos, M.; Deri, L. pcapIndex: An Index for Network Packet Traces with Legacy Compatibility. *ACM SIGCOMM Comput. Commun. Rev.* **2012**, *42*, 47–53. [CrossRef]
14. Ponec, M.; Giura, P.; Wein, J.; Bronnimann, H. New Payload Attribution Methods for Network Forensic Investigations. *ACM Trans. Inf. Syst. Secur.* **2010**, *13*, 15. [CrossRef]
15. Giura, P.; Memon, N. NetStore: An Efficient Storage Infrastructure for Network Forensics and Monitoring. In *Recent Advances in Intrusion Detection (RAID 2010)*; Lecture Notes in Computer Science; Springer: Berlin/Heidelberg, Germany, 2010; pp. 277–296. [CrossRef]
16. Oliveira, R.; Pedrosa, T.; Rufino, J.; Lopes, R.P. Parameterization and Performance Analysis of a Scalable, near Real-Time Packet Capturing Platform. *Systems* **2024**, *12*, 126. [CrossRef]
17. Song, W.; Beshley, M.; Przystupa, K.; Beshley, H.; Kochan, O.; Pryslupskyi, A.; Pieniak, D.; Su, J. A Software Deep Packet Inspection System for Network Traffic Analysis and Anomaly Detection. *Sensors* **2020**, *20*, 1637. [CrossRef] [PubMed]
18. Lashkari, A.H.; Draper-Gil, G.; Mamun, M.S.I.; Ghorbani, A.A. Characterization of Tor Traffic Using Time Based Features. In Proceedings of the 3rd International Conference on Information Systems Security and Privacy (ICISSP 2017), Porto, Portugal, 19–21 February 2017; pp. 253–262. [CrossRef]
19. Draper-Gil, G.; Lashkari, A.H.; Mamun, M.S.I.; Ghorbani, A.A. Characterization of Encrypted and VPN Traffic Using Time-Related Features. In Proceedings of the 2nd International Conference on Information Systems Security and Privacy (ICISSP 2016), Rome, Italy, 19–21 February 2016; pp. 407–414. [CrossRef]
20. Sharafaldin, I.; Lashkari, A.H.; Ghorbani, A.A. Toward Generating a New Intrusion Detection Dataset and Intrusion Traffic Characterization. In Proceedings of the 4th International Conference on Information Systems Security and Privacy (ICISSP), Funchal, Portugal, 22–24 January 2018; pp. 108–116. [CrossRef]
21. ntop. nDPI: Open Source Deep Packet Inspection Library. Available online: https://www.ntop.org/products/deep-packet-inspection/ndpi/ (accessed on 24 August 2026).
22. Cisco Systems. Joy: A Package for Capturing and Analyzing Network Flow Data. Available online: https://github.com/cisco/joy (accessed on 24 August 2026).
23. Yu, H.; Yang, W.; Cui, B.; Sui, R.; Wu, X. Renyi Entropy-Driven Network Traffic Anomaly Detection with Dynamic Threshold. *Cybersecurity* **2024**, *7*, 64. [CrossRef]
24. Abu Bakar, R.; Kijsirikul, B. Enhancing Network Visibility and Security with Advanced Port Scanning Techniques. *Sensors* **2023**, *23*, 7541. [CrossRef] [PubMed]
25. Ring, M.; Landes, D.; Hotho, A. Detection of Slow Port Scans in Flow-Based Network Traffic. *PLoS ONE* **2018**, *13*, e0204507. [CrossRef] [PubMed]
26. Sikora, M.; Gerlich, T.; Malina, L. On Detection and Mitigation of Slow Rate Denial of Service Attacks. In Proceedings of the 11th International Congress on Ultra Modern Telecommunications and Control Systems and Workshops (ICUMT), Dublin, Ireland, 28–30 October 2019; pp. 1–5. [CrossRef]
27. Vedula, V.; Lama, P.; Boppana, R.V.; Trejo, L.A. On the Detection of Low-Rate Denial of Service Attacks at Transport and Application Layers. *Electronics* **2021**, *10*, 2105. [CrossRef]
28. Aouini, Z.; Pekar, A. NFStream: A Flexible Network Data Analysis Framework. *Comput. Netw.* **2022**, *204*, 108719. [CrossRef]
29. Burschka, S.; Dupasquier, B. Tranalyzer: Versatile High Performance Network Traffic Analyser. In Proceedings of the 2016 IEEE Symposium Series on Computational Intelligence (SSCI), Athens, Greece, 6–9 December 2016; pp. 1–8. [CrossRef]
30. Shafi, M.; Lashkari, A.H.; Roudsari, A.H. NTLFlowLyzer: Towards Generating an Intrusion Detection Dataset and Intruders Behavior Profiling through Network and Transport Layers Traffic Analysis and Pattern Extraction. *Comput. Secur.* **2025**, *148*, 104160. [CrossRef]
31. Holland, J.; Schmitt, P.; Feamster, N.; Mittal, P. New Directions in Automated Traffic Analysis. In Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security (CCS '21), Virtual Event, 15–19 November 2021; pp. 3366–3383. [CrossRef]
32. Rescorla, E. *The Transport Layer Security (TLS) Protocol Version 1.3*; RFC 9846; RFC Editor: Wilmington, DE, USA, 2026. Available online: https://www.rfc-editor.org/rfc/rfc9846.html (accessed on 24 August 2026).
33. Mockapetris, P. *Domain Names—Implementation and Specification*; RFC 1035; RFC Editor: Wilmington, DE, USA, 1987. [CrossRef]
34. Fielding, R.; Nottingham, M.; Reschke, J. *HTTP Semantics*; RFC 9110; RFC Editor: Wilmington, DE, USA, 2022. [CrossRef]
35. Fielding, R.; Nottingham, M.; Reschke, J. (Eds.) *HTTP/1.1*; RFC 9112; RFC Editor: Wilmington, DE, USA, 2022. [CrossRef]
36. Harris, G.; Richardson, M. (Eds.) *PCAP Capture File Format*; Internet-Draft draft-ietf-opsawg-pcap-08; Internet Engineering Task Force: Wilmington, DE, USA, 2026. Available online: https://datatracker.ietf.org/doc/draft-ietf-opsawg-pcap/ (accessed on 24 August 2026).

37. Tüxen, M.; Risso, F.; Bongertz, J.; Combs, G.; Harris, G.; Chaudron, E.; Richardson, M. (Eds.) *PCAP Now Generic (pcapng) Capture File Format*; Internet-Draft draft-ietf-opsawg-pcapng-05; Internet Engineering Task Force: Wilmington, DE, USA, 2026. Available online: https://datatracker.ietf.org/doc/draft-ietf-opsawg-pcapng/ (accessed on 24 August 2026).
38. Braun, L.; Didebulidze, A.; Kammenhuber, N.; Carle, G. Comparing and Improving Current Packet Capturing Solutions Based on Commodity Hardware. In Proceedings of the 10th ACM SIGCOMM Conference on Internet Measurement (IMC), Melbourne, Australia, 1–3 November 2010; pp. 206–217. [CrossRef]
39. Rizzo, L. Revisiting Network I/O APIs: The netmap Framework. *ACM Queue* **2012**, *10*, 30–39. [CrossRef]
40. Postel, J. *Internet Control Message Protocol*; RFC 792; RFC Editor: Wilmington, DE, USA, 1981. [CrossRef]
41. Eddy, W. *Transmission Control Protocol (TCP)*; RFC 9293; RFC Editor: Wilmington, DE, USA, 2022. [CrossRef]
42. Zhou, J.; Fu, W.; Hu, W.; Sun, Z.; He, T.; Zhang, Z. Challenges and Advances in Analyzing TLS 1.3-Encrypted Traffic: A Comprehensive Survey. *Electronics* **2024**, *13*, 4000. [CrossRef]
43. Jung, I.; Song, Y.; Jilcha, L.; Kim, D.; Im, S.; Shim, S.; Kim, Y.; Kwak, J. Enhanced Encrypted Traffic Analysis Leveraging Graph Neural Networks and Optimized Feature Dimensionality Reduction. *Symmetry* **2024**, *16*, 733. [CrossRef]
44. Papadogiannaki, E.; Ioannidis, S. A Survey on Encrypted Network Traffic Analysis Applications, Techniques, and Countermeasures. *ACM Comput. Surv.* **2021**, *54*, 1–35. [CrossRef]
45. Salat, L.; Davis, M.; Khan, N. DNS Tunnelling, Exfiltration and Detection over Cloud Environments. *Sensors* **2023**, *23*, 2760. [CrossRef] [PubMed]
46. Plohmann, D.; Yakdan, K.; Klatt, M.; Bader, J.; Gerhards-Padilla, E. A Comprehensive Measurement Study of Domain Generating Malware. In Proceedings of the 25th USENIX Security Symposium, Austin, TX, USA, 10–12 August 2016; pp. 263–278. Available online: https://www.usenix.org/conference/usenixsecurity16/technical-sessions/presentation/plohmann (accessed on 25 August 2026).
47. Shannon, C.E. A Mathematical Theory of Communication. *Bell Syst. Tech. J.* **1948**, *27*, 379–423. [CrossRef]
48. Kenyon, A.; Deka, L.; Elizondo, D. Characterising Payload Entropy in Packet Flows—Baseline Entropy Analysis for Network Anomaly Detection. *Future Internet* **2024**, *16*, 470. [CrossRef]
49. Kilincer, I.F.; Ertam, F.; Sengur, A. Machine Learning Methods for Cyber Security Intrusion Detection: Datasets and Comparative Study. *Comput. Netw.* **2021**, *188*, 107840. [CrossRef]
50. Ferriyan, A.; Thamrin, A.H.; Takeda, K.; Murai, J. Generating Network Intrusion Detection Dataset Based on Real and Encrypted Synthetic Attack Traffic. *Appl. Sci.* **2021**, *11*, 7868. [CrossRef]
51. Sharafaldin, I.; Lashkari, A.H.; Hakak, S.; Ghorbani, A.A. Developing Realistic Distributed Denial of Service (DDoS) Attack Dataset and Taxonomy. In Proceedings of the 2019 International Carnahan Conference on Security Technology (ICCST), Chennai, India, 1–3 October 2019; pp. 1–8. [CrossRef]
52. Canadian Institute for Cybersecurity; Communications Security Establishment. CSE-CIC-IDS2018: A Realistic Cyber Defense Dataset. Available online: https://www.unb.ca/cic/datasets/ids-2018.html (accessed on 29 September 2026).
53. Eddy, W. *TCP SYN Flooding Attacks and Common Mitigations*; RFC 4987; RFC Editor: Wilmington, DE, USA, 2007. [CrossRef]
54. Yang, C.-H.; Wu, J.-P.; Lee, F.-Y.; Lin, T.-Y.; Tsai, M.-H. Detection and Mitigation of SYN Flooding Attacks through SYN/ACK Packets and Black/White Lists. *Sensors* **2023**, *23*, 3817. [CrossRef] [PubMed]
55. Engelen, G.; Rimmer, V.; Joosen, W. Troubleshooting an Intrusion Detection Dataset: The CICIDS2017 Case Study. In Proceedings of the 2021 IEEE Security and Privacy Workshops (SPW), San Francisco, CA, USA, 23–27 May 2021; pp. 7–12. [CrossRef]
56. Lanvin, M.; Gimenez, P.-F.; Han, Y.; Majorczyk, F.; Mé, L.; Totel, É. Errors in the CICIDS2017 Dataset and the Significant Differences in Detection Performances It Makes. In *Risks and Security of Internet and Systems (CRiSIS 2022)*; Lecture Notes in Computer Science; Springer: Berlin/Heidelberg, Germany, 2023; pp. 18–33. [CrossRef]
57. Lypa, B.; Horyn, I.; Zagorodna, N.; Tymoshchuk, D.; Lechachenko, T. Comparison of feature extraction tools for network traffic data. In Proceedings of the 4th International Workshop on Information Technologies: Theoretical and Applied Problems (ITTAP 2024), Ternopil, Ukraine, 23–25 October 2024. [CrossRef]
58. Lashkari, A.H. CICFlowMeter V4.0. Available online: https://github.com/ahlashkari/CICFlowMeter (accessed on 25 August 2026).

Disclaimer/Publisher's Note: The statements, opinions and data contained in all publications are solely those of the individual author(s) and contributor(s) and not of MDPI and/or the editor(s). MDPI and/or the editor(s) disclaim responsibility for any injury to people or property resulting from any ideas, methods, instructions or products referred to in the content.