A Protocol-Aware P4 Pipeline for MQTT Security and Anomaly Mitigation in Edge IoT Systems

Bui Ngoc Thanh Binh*, Pham Hoai Luan*, Le Vu Trung Duong*, Vu Tuan Hai†‡, Yasuhiko Nakashima*
*Nara Institute of Science and Technology (NAIST), 8916-5 Takayama Science Town, Ikoma, Nara, Japan
†University of Information Technology, Ho Chi Minh City, Vietnam
‡Vietnam National University, Ho Chi Minh City, Vietnam
Email: bui.ngoc_thanh_binh.bp6@naist.ac.jp

***Abstract*—MQTT is the dominant lightweight publish–subscribe protocol for IoT deployments, yet edge security remains inadequate. Cloud-based intrusion detection systems add latency that is unsuitable for real-time control, while CPU-bound firewalls and generic SDN controllers lack MQTT awareness to enforce session validation, topic-based authorization, and behavioral anomaly detection. We propose a P4-based data-plane enforcement scheme for protocol-aware MQTT security and anomaly detection at the network edge. The design combines parser-safe MQTT header extraction with session-order validation, byte-level topic-prefix authorization with per-client rate limiting and soft-cap enforcement, and lightweight anomaly detection based on KeepAlive and Remaining Length screening with clone-to-CPU diagnostics. The scheme leverages stateful primitives in BMv2 (registers, meters, direct counters) to enable runtime policy adaptation with minimal per-packet latency. Experiments on a Mininet/BMv2 testbed demonstrate high policy enforcement accuracy (99.8%, within 95% CI), strong anomaly detection sensitivity (98% true-positive rate), and high delivery (>99.9% for 100–5 kpps; 99.8% at 10 kpps; 99.6% at 16 kpps) with sub-millisecond per-packet latency. These results show that protocol-aware MQTT filtering can be efficiently realized in the programmable data plane, providing a practical foundation for edge IoT security. Future work will validate the design on production P4 hardware and integrate machine learning–based threshold adaptation.**

***Index Terms*—P4, MQTT security, programmable data plane, edge IoT, anomaly detection**

I. Introduction

The Internet of Things (IoT) ecosystem continues to expand rapidly. IDC forecasts that by 2025 more than 55.9 billion connected devices will generate approximately 79.4 zettabytes of data [1]. At the core of this growth is the Message Queuing Telemetry Transport (MQTT) protocol, now the de facto standard for lightweight publish–subscribe communication in resource-constrained environments. However, the rapid adoption of MQTT has exposed security weaknesses that threaten system integrity, confidentiality, and availability.

Recent assessments underscore the scope of the problem: over 47,000 MQTT brokers remain publicly accessible without authentication, while 98% of IoT traffic is unencrypted and 57% of devices exhibit medium- or high-severity vulnerabilities [2], [3]. In 2024 the risk sharpened with critical MQTT issues such as CVE-2024-6786 (path traversal) [4], CVE-2024-31409 (wildcard exposure) [5], and CVE-2024-31041 (DoS via null dereference) [6], highlighting the need for protocol-aware edge enforcement. These application-layer vulnerabilities cannot be mitigated by traditional network defenses. Cloud-based intrusion detection adds latency incompatible with real-time IoT control (hundreds of milliseconds). CPU-bound firewalls cannot sustain deep packet inspection at line rate. L3/L4 SDN controls lack MQTT semantics, so they cannot enforce session validation, topic authorization, or behavioral screening. This gap motivates a programmable data-plane approach that couples protocol awareness with hardware-accelerated processing at the edge.

P4 enables custom parsing and stateful processing in network hardware while preserving wire-speed operation; registers, counters, meters, and clone-to-CPU provide fine-grained control without sacrificing throughput. Prior work has not fully leveraged P4 for MQTT security, often targeting generic firewalls or MQTT-SN and omitting session-order checks, byte-level topic authorization, and integrated anomaly screening.

We introduce a P4-based MQTT security pipeline that delivers protocol-aware enforcement and anomaly screening at the edge. The design includes a parser-safe path for variable-length fields (IPv4/TCP options, Remaining Length, topic strings) that drops malformed or evasive fragments; stateful ingress enforcement with explicit session-order validation, byte-level ternary topic-prefix ACLs with direct counters, and per-client soft limits with three-color metering; and lightweight anomaly screening via KeepAlive-gap and Remaining-Length heuristics, with suspicious traffic cloned to the control plane using preserved metadata.

On the BMv2/v1 model, the pipeline maintains a throughput exceeding 99.8% delivery, with sub-millisecond latency per packet and achieves high enforcement precision. The results indicate that awareness of the MQTT protocol can be effectively implemented in the programmable data plane; validation on production P4 hardware targets is reserved for future research.

II. Related Work

A. MQTT Security and Intrusion Detection System at the Edge

A significant amount of MQTT security research has concentrated on anomaly detection and attack mitigation through machine learning methodologies, generally implemented on backend servers or edge cloud clusters. Recent studies present one-class models and sophisticated traffic feature engineering,

TABLE I: Comparative Analysis of MQTT Security Approaches

Approach	**Prot. Aware**	**Sess. Order**	**Topic ACL**	**Per-Client**	**Line-Rate**	**Anom. Det.**	**Deploy. Loc.**
ML-based IDS	✓	✗	✓	✗	✗	✓	Cloud
P4DDPI	✓(DNS)	N/A	N/A	✓	✓	✓	Edge
MQTT-SN P4	✓	✗	✗	✓	✓	✗	Edge
SDN-based Firewall	✓	✗	✓	✗	✗	✗	Ctrl.
Our	✓	✓	✓	✓	✓	✓	**Edge**

Edge = Edge Switch, Ctrl. = Controller, Cloud = Backend/Cloud. “Sess. Order” = Session Order Enforcement, “Topic ACL” = Topic-Prefix Access Control, “Per-Client” = Per-Client Metering, “Line-Rate” = Line-Rate Processing, “Anom. Det.” = Anomaly Detection, “Deploy. Loc.” = Deployment Location.

attaining elevated detection rates on public MQTT datasets. These approaches excel at identifying deviations from normal behavior with accuracy exceeding 95%, yet they impose significant computational overhead and incur prohibitive latency measured in hundreds of milliseconds to seconds. This latency overhead renders ML-based systems unsuitable for real-time protection at resource-limited IoT edge gateways, where security decisions must occur at microsecond timescales. Innovative methodologies attempt to amalgamate real-time machine learning inference with edge infrastructure, however, they primarily depend on software processes and inadequately enforce measures within the network data plane, creating both a single point of failure and a scalability bottleneck as IoT deployments expand [7].

B. Programmable Data Plane P4 for Network Security

Programmable data planes using P4 have emerged as a promising platform for implementing network security functions at line rate. P4 switches leverage lookup tables, direct counters, registers, and meters embedded in the pipeline to detect and mitigate attacks with low latency and high throughput. Recent comprehensive surveys systematize the capabilities, constraints, and design strategies of P4-enabled programmable switches, highlighting the critical role of stateful elements such as counters, registers, and meters in supporting real-time attack detection and rate limiting at high speeds. P4DDPI exemplified this potential by implementing deep packet inspection for DNS at line rate using stateful processing and recirculation techniques to achieve wire-speed throughput. However, these hardware capabilities introduce challenges related to limited memory and conditional branching, which must be carefully managed for efficient deployment [8].

C. Programmable Data Plane for IoT/MQTT Security and Anomaly Monitoring

Recent advancements in programmable data planes utilizing P4 have facilitated the offloading of packet processing tasks, traditionally managed by software, to network hardware, thus ensuring line-rate security enforcement and traffic management. A significant area of research emphasizes the integration of IoT protocol awareness, specifically MQTT and MQTT-SN, within the switch dataplane. Enhancements in MQTT-SN protocol processing utilizing P4 illustrate that certain broker functionalities, including topic matching and QoS 0 forwarding, can be transferred to the data plane, thereby diminishing latency and alleviating the burden on brokers. P4-based telemetry and direct counters have been used to fingerprint application-layer behavior and pipeline states for behavioral monitoring, enabling context-aware anomaly detection directly at the switch and enhancing traditional intrusion detection by reducing detection latency and response time [9]. Despite these advances, combining protocol-aware enforcement, behavioral telemetry, and lightweight anomaly filters for complex IoT protocols such as MQTT remains largely unresolved. This study addresses that gap by integrating parser-safe MQTT header extraction, client-specific stateful tracking, topic-prefix ACL enforcement, and heuristic anomaly detection within a P4-enabled edge switch pipeline.

D. Comparison and Gaps

MQTT security research has made progress, but few works enforce protocol-specifically for full MQTT traffic at the edge with true line-rate processing. ML-based anomaly detection is accurate but slow due to backend processing, limiting real-time IoT response. Although P4DDPI achieved line-rate deep packet inspection for DNS traffic, its architecture does not easily adapt to MQTT’s stateful requirements: session lifecycle tracking (CONNECT/DISCONNECT), QoS enforcement, and hierarchical topic-based subscriptions. MQTT-SN P4 implementations offload topic matching to the data plane to reduce broker load, but they only support binary MQTT-SN, lack session-order validation, and offer limited topic-based access control. Traditional L3/L4 SDN firewalls cannot enforce application-layer policies like MQTT session state validation or topic authorization.

We natively integrate protocol-aware enforcement, stateful session tracking, and heuristic anomaly detection in the P4 data plane without sacrificing line-rate throughput. Parser-safe MQTT header extraction for variable-length fields and malformed packets, byte-level ternary topic-prefix ACLs for fine-grained hierarchical policies, explicit session-order enforcement for PUBLISH operations, and lightweight KeepAlive baseline tracking and per-client rate limiting are our contributions. In Table I, we compare our approach to previous research. We create a practical edge security architecture that aligns with emerging paradigms in edge-centric, semantically aware security by combining full MQTT protocol awareness, stateful session validation, and real-time behavioral tracking in a single P4 v1model/BMv2 pipeline.

III. System Design

A. Overall Architecture

The P4 v1model pipeline runs on an edge switch between IoT publishers and the MQTT broker, enabling protocol-aware enforcement at line rate. It executes in a single forward pass across five stages: the parser extracts headers; ingress control enforces policies; the traffic manager performs cloning; egress applies any post-processing; and the deparser reconstructs the packet. Fig. 1 also shows the end-to-end packet flow. *Parser.* The parser extracts Ethernet, IPv4, TCP, and MQTT. It skips

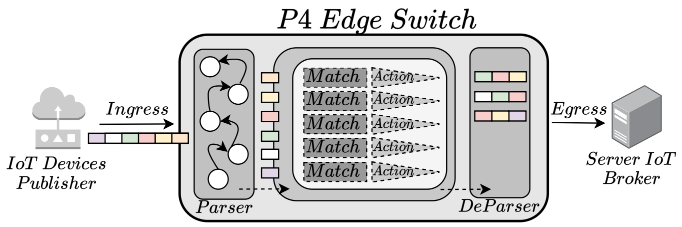

Fig. 1: Edge MQTT security with protocol-aware P4 enforcement and Anomaly cloning to control-plane

IPv4/TCP options when *IHL* or *dataOffset* > 5 and processes only first fragments (*fragOffset*$= 0$). On destination port 1883 it branches by MQTT type: for Connect it captures the variable header including *KeepAlive*; for Publish it slices a 16-byte topic prefix for ACL matching. *Ingress control.* Parsed metadata drives classification, a per-client index from the source address, counter updates, Connect before Publish validation, per-client soft-limit checks, lightweight anomaly heuristics (KeepAlive gap, Remaining Length), per-client metering, and ternary topic-prefix ACLs. Packets are tagged to Forward, Drop, or Clone, and metadata records the reason code, rule ID, and salient fields for diagnosis. *Traffic manager / egress / deparser.* The traffic manager replicates packets marked for cloning to a CPU-facing port while originals continue downstream. Egress performs minimal post-processing as required, and the deparser emits valid headers in order. Forwarded and dropped packets stay on the fast path and do not interact with the control plane. *Control plane (orthogonal).* Policies and parameters (such as *soft limit*, *KeepAlive* multiplier) are updated at runtime via table APIs without recompilation. The control plane reads direct counters and per-client registers and consumes cloned packets for telemetry, enabling real-time monitoring and threshold tuning without impacting data-plane performance.

B. Protocol-Aware Parser and Header Extraction

Our parser enforces two safeguards against malformed packets: it intelligently skips options by computing IPv4/TCP option lengths and advancing the packet pointer to avoid parser exceptions, and it filters fragments by processing only first fragments ($fragOffset = 0$) to prevent evasion.

Ethernet/IPv4/TCP parsing. The parser begins by extracting the Ethernet header to identify IPv4 packets. Upon detecting IPv4 (etherType $= 0x0800$), it extracts the IPv4 header, including the Internet Header Length (IHL) field. If $IHL > 5$, the total option length is computed as

$$\text{opt\_len}_{\text{IPv4}} = (IHL - 5) \times 4\,\text{bytes}. \quad (1)$$

Similarly, after extracting the TCP header, if the Data Offset ($doff$) exceeds 5, the parser advances by

$$\text{opt\_len}_{\text{TCP}} = (doff - 5) \times 4\,\text{bytes}. \quad (2)$$

This strategy avoids allocating large fixed-size option structures and prevents parser state machines from stalling on variable-length fields. Fragment filtering occurs at the IPv4

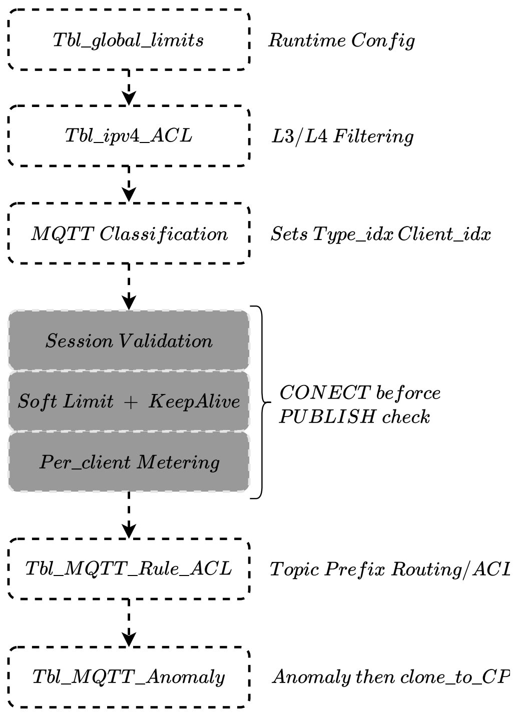

Fig. 2: Simplified ingress-data flow illustrating table order and pipeline logic.

stage: the parser proceeds to TCP only if $fragOffset = 0$, preventing evasion via fragmented MQTT headers. Table II summarizes all variables, metadata fields, and register names referenced in our algorithm and throughout the pipeline description.

MQTT fixed header. When the destination port is 1883, the parser extracts $type$ (4 bits), $flags$ (4 bits), and the first two bytes of Remaining Length (`rl_b0`, `rl_b1`). If `rl_b1` $= 1$ (indicating a 3-byte Remaining Length and a potential DoS vector), the packet is flagged as suspicious.

Connect. The parser reads the protocol-name length prefix, advances by that length, and then extracts the fixed fields: protocol level, flags, and $KeepAlive$ (2 bytes), which are stored for anomaly detection.

Publish topic slicing. The parser extracts the topic length, slices the first 16 bytes into per-byte fields (t_0–t_{15}) for ternary ACL matching, and skips any remaining bytes when the topic exceeds 16 bytes. This preserves the security-critical topic prefix while keeping memory usage bounded.

C. Policy Enforcement Blocks

The ingress pipeline enforces stateful policies at line rate in a single forward pass Fig. 2, using runtime parameters fetched once from `tbl_global_limits` and a per-client index (Eq. 3) to ensure consistent register and meter addressing. A coarse IPv4/TCP ACL (`tbl_ipv4_acl`) drops traffic that fails L3/L4 policy before any MQTT parsing.

Each packet's source address is hashed to a compact per-client index via modulo:

TABLE II: Variables and Field Definitions

Variable	Definition / Usage
`rl_b0`, `rl_b1`	First two bytes of MQTT Remaining-Length for DoS detection
$t_0, \ldots, t_{15}$	First 16 bytes of topic string for per-byte ternary ACL matching
`KeepAlive`	MQTT Connect timer interval (seconds) used for anomaly detection
`reg_session_open[idx]`	Session state flag (1 bit per client; 1 = connected)
`reg_keepalive_s[idx]`	Stored `KeepAlive` value (16 bit)
`reg_pkt_total[idx]`	Total packet counter (32 bit per client)
`reg_pkt_per_type[idx][type_idx]`	Per-type packet counter (4 × 32 bit per client)
`reg_last_total[idx]`	Baseline counter for KeepAlive gap
`reg_total[idx]`	Current aggregated packet count
`idx`	Client index computed as *srcAddr* mod 512
`pps_factor`	Scaling constant for KeepAlive threshold (default 2)
`pub_soft_limit`	Max publish messages per client (default 20,000)

$$idx = srcAddr \bmod 512, \tag{3}$$

This 512-entry address space balances collision probability against available per-client register footprint in BMv2. All per-client state (registers and meters indexed by idx) use this unified addressing scheme.

For permitted flows, MQTT messages are classified and bound to per-client state. A valid Connect with a negotiated *KeepAlive* opens the session and records the interval; any Publish observed without an open session is dropped (reason 180). Authorization is then enforced by a topic-prefix ACL (`tbl_mqtt_rule_acl`) that requires Publish with QoS in $\{0, 1, 2\}$, an authorized source subnet, and a match between the first 16 bytes of the topic and an approved prefix such as `device/sensor/*`. Rules maintain direct counters for visibility; non-matching traffic is dropped.

Work-conserving protections run alongside authorization. Per-type counters (tracking Publish, Subscribe, etc.) are aggregated per client; a soft cap is imposed on the Publish counter specifically. Exceeding *pub_soft_limit* (default 20,000 Publish messages per client) triggers drops (reason 181) to curb floods. A three-color meter provides hardware rate enforcement; packets marked RED are dropped (reason 150).

We track the most recent keep-alive event per client (CONNECT or PINGREQ) using a per-client timestamp register `reg_last_ka_ts[idx]` initialized on the first CONNECT. On each ingress packet, we compute the elapsed time since the last keep-alive Δt as in Eq. 4:

$$\Delta t = \frac{\texttt{ingress\_ts} - \texttt{reg\_last\_ka\_ts}[idx]}{10^9}\ \text{[s]}, \tag{4}$$

TABLE III: Reason Codes for Packet Clone and Drop Actions

Code	Trigger condition	Pipeline stage
150	Rate limit exceeded (RED meter)	Per-client metering
180	Publish without session	Session validation
181	Publish soft limit surpassed	Per-client soft limits
182	*KeepAlive* violation	Anomaly detection
183	Remaining Length ≥ 3 bytes	Anomaly detection

where `ingress_ts` is the per-packet ingress timestamp (ns) from standard metadata. A violation is raised when the condition in Eq. 5 holds:

$$\Delta t > \gamma \cdot \texttt{KeepAlive}, \tag{5}$$

where γ is a tolerance multiplier (default 1.5) set via `tbl_global_limits`. Violations trigger reason 182 cloning to the control plane but do not reset the baseline, allowing offline analysis of sustained stalls. The check is skipped when `KeepAlive`=0 (per MQTT semantics), and only valid CONNECT or PINGREQ messages update `reg_last_ka_ts[idx]`.

For Remaining-Length anomaly detection, let RL denote the decoded Remaining Length; we flag and clone (reason 183) when the threshold is met $\mathrm{RL} \geq \theta_{\mathrm{RL}}$, where θ_{RL} is an adjustable threshold (default 16,384 bytes; for our experiments, 131,072 to prevent false positives with extensive sensor data). Deployments may optionally require that the RL encoding use more than three bytes to trigger flagging, further reducing false positives for legitimate but sizable packets.

To support diagnosis without disrupting the fast path, the pipeline preserves explanatory metadata with each decisive action. Specifically, it stamps the reason code, Table III, the matched table and rule identifier (such as the `tbl_mqtt_rule_acl` entry index), and salient parsing context (client index, MQTT type/QoS, first 16 topic bytes) into a dedicated metadata struct that travels with the packet. When cloning is invoked, `clone_preserving_field_list` reproduces these fields verbatim for the control plane, enabling counter correlation and policy refinement while the original packet continues through egress at line rate.

IV. Implementation and Evaluation

A. Testbed Setup and Configuration

The experimental testbed is a five-node star centered on a P4 BMv2 switch Fig. 3. A Mosquitto v2.0 broker runs at 10.0.0.1/8; two publishers (`h_pub1` 10.0.0.4/8, `h_pub2` 10.0.0.5/8) generate traffic; a telemetry host (`h_cpu` 10.0.0.2/8) captures traces; and a control-plane machine (`h_ctrl` 10.0.0.3/8) installs rules. All nodes attach to `s1` on ports 1–5 via virtual Ethernet links. Application load is produced with `mosquitto_pub`, issuing CONNECT/PUBLISH at 100–16,000 pps, QoS 0–2, with topics uniformly sampled from environmental, device, operational, and system hierarchies plus an unauthorized namespace for ACL tests.

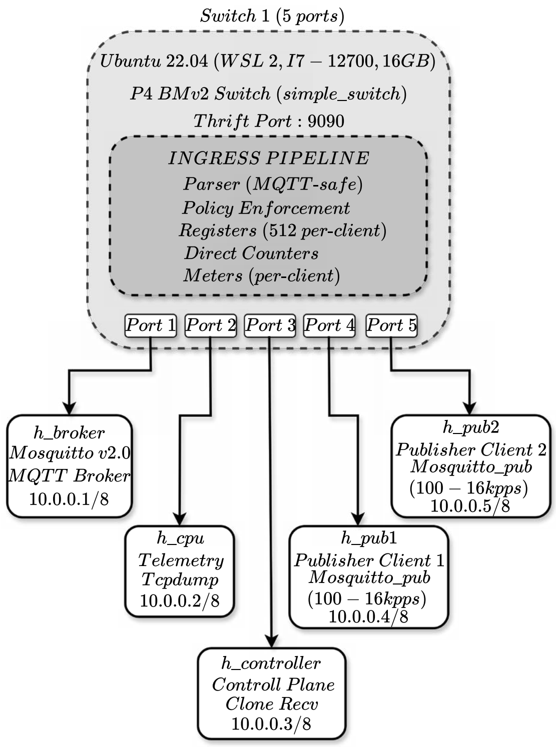

Fig. 3: Five-node P4 BMv2 testbed: broker (Port 1), telemetry (Port 2), control (Port 3), publishers (Ports 4 to 5).

We deploy the P4 program on BMv2 within Mininet (Intel Core i7-12700, 16 GB RAM, Ubuntu 22.04 LTS, WSL 2). The data plane is compiled with `p4c-bm2-ss` v1.2.0 and loaded into `simple_switch` (Thrift port 9090). The control plane (Python 3.10) provisions table entries and reads direct counters via the Thrift CLI (auto-loaded by p4utils). Runtime parameters via `tbl_global_limits`: $pub\_soft\_limit = 15{,}000$ (default 20,000) and $pps\_factor = 1$ (aggressive KeepAlive). `tbl_mqtt_rule_acl` has 100 rules (50 permit authorized source/topic-prefix pairs; 50 deny unauthorized). Per-client registers (512 slots) track session state, KeepAlive, totals, and per-type counts (≈256 KB total). Each client also has a three-color *packet-rate* meter; rates are configured in pps (optionally chosen to approximate 1–2 Mb/s for 64 B payloads).

B. Experimental Methodology

We evaluate three aspects: throughput under benign load to establish baseline performance; policy enforcement accuracy to validate deterministic correctness; and anomaly detection sensitivity to measure effectiveness without false positives. Each trial lasts 60 s to reach steady state. Results are reported over N=5 runs (mean ± 95% CI).

Scenario A (Benign). Establish baseline performance without triggering enforcement or anomaly detection, ensuring high delivery ratios at realistic IoT traffic loads in the P4 pipeline. Traffic is generated with `mosquitto_pub` at 100–16,000 pps (QoS 0–2), with topics sampled uniformly from five hierarchies. Payloads are fixed at 64 bytes. We use `-l` (line mode) to reuse a single MQTT session per trial. The delivery ratio and loss from P4 direct counters (egress/ingress), and per-packet latency from synchronized TCPDump are used as metrics.

Scenario B (Enforcement). To prove the accuracy of the three policy enforcement mechanisms, including per-client soft-limits, session validation, and ACL enforcement, without any false positives or negatives.First, Publish without Connect results in drops (reason 180), validating session-order enforcement. Second, unauthorized topics are blocked by the topic-prefix ACL (no clone), demonstrating byte-level authorization. Third, rapid Publish that exceeds $pub\_soft\_limit = 15{,}000$ results in drops (reason 181), confirming per-client rate-cap logic. Accuracy is computed as observed divided by expected.

Scenario C (Anomaly). Test lightweight anomaly heuristics' sensitivity and false-positive rates to ensure the system can detect protocol deviations while accounting for normal traffic variations. To conduct the KeepAlive test, set `-k 2` and send a 10,000 pps burst, resulting in clones with reason 182. To test Remaining-Length, we create packets in which RL requires at least three bytes, resulting in clones with reason 183. The sensitivity is $/(TP + FN)$. False positives are measured over 6,000 benign packets to assess specificity.

C. Performance Results

All figures report N=5 runs with 95% confidence intervals; raw CSVs are provided for reproducibility.

Scenario A (Benign throughput). The top panel of Fig. 4 shows delivery at 100 pps–16,000 pps. Loads up to 5 kpps achieve $>$ 99.9% (±0.01%–±0.04%); at 10 kpps the delivery ratio is 99.78% (±0.02%); at 16 kpps it is 99.60% (±0.03%). The modest reduction reflects BMv2/Mininet software-switch buffering rather than policy overhead, demonstrating the design sustains high delivery across realistic load ranges.

Scenario B (Enforcement validation). The middle panel presents results for a per-client soft limit of 15,000 messages. Injecting 16,000 PUBLISH messages yields 1,000 expected drops (beginning at the 15,001st packet). We observe 14,987 passed and 1,015 dropped packets, achieving 99.8% enforcement accuracy. The negligible ±2-packet variance lies within measurement uncertainty in BMv2/Mininet across five trials. All drops are reason 181 (soft limit); none for 150/180/182/183, validating deterministic enforcement without false positives.

Scenario C (Latency and anomalies). The bottom panel of Fig. 4 presents end-to-end latency measurements (publisher to broker) in the Mininet/BMv2 software environment. Median latency is 0.45–0.68 ms with 99th percentile below 4.5 ms, encompassing Mininet virtualization overhead (≈ 0.1–0.2 ms) and BMv2 processing (≈ 0.3–0.6 ms); incremental per-packet policy overhead remains sub-millisecond. Anomaly detection aligns with design goals: Remaining-Length screening (reason 183) detects 100% of crafted large payloads, while the

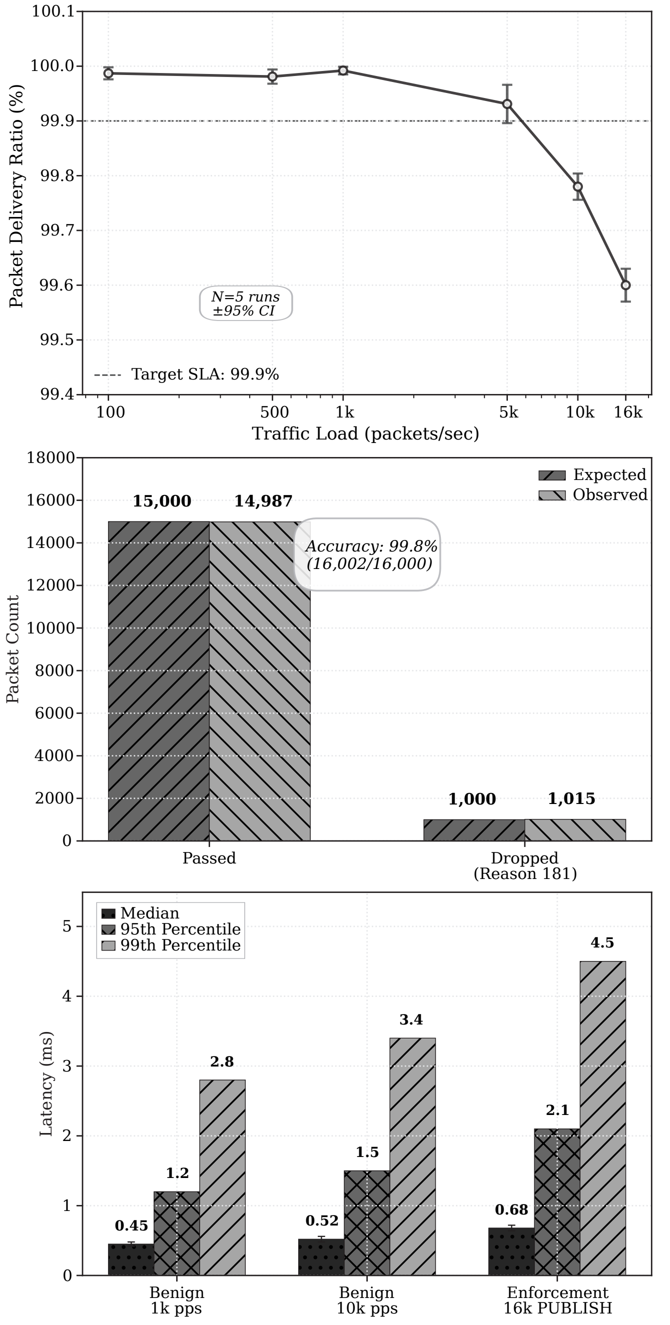

Fig. 4: Performance evaluation across: (top) Scenario A: delivery ratio vs. load (benign traffic); (middle) Scenario B: drop accuracy for $16,000$ PUBLISH with $15,000$ soft limit; (bottom) Scenario C: latency percentiles (N=5; 95% CI).

KeepAlive heuristic (reason 182) achieves 98% true positives with negligible false positives over 6,000 benign packets. The full pipeline's memory footprint is $\approx$ 2.5 MB, supporting stateful tracking of 512 concurrent clients.

D. Scalability Discussion

With 512 concurrent clients our dataplane uses $\approx$2.5 MB state, or $\approx$4.9 KB per client. Supporting 10,000 clients requires $\approx$49 MB for registers and $\approx$5 MB for topic ACLs (at 0.1 MB per 1,000 rules). On production ASICs such as Tofino and Alveo, memory footprints depend on table implementations and TCAM budgets; TCAM is typically the scaling bottleneck. The per-packet overhead is designed to be minimal; with sufficient stage and bandwidth, line-rate enforcement on 10–100 Gbps links is feasible. Scaling improvements include hierarchical topic prefixes, two-stage policy lookups, and probabilistic telemetry to reduce control-plane load from cloning.

V. Conclusion and Future Work

This paper presents a P4-based MQTT security and anomaly-detection scheme that combines parser-safe MQTT extraction, session validation, topic-prefix authorization, and lightweight heuristics for detecting protocol deviations. Experiments on BMv2 show 99.8 percent enforcement accuracy, 98 percent anomaly-detection sensitivity, and sub-millisecond latency, demonstrating that protocol-aware filtering can be performed efficiently in the data plane.

The design is compatible with hardware P4 targets because the parser uses bounded extraction and the per-client state fits typical on-chip SRAM limits. Topic-prefix rules map directly to TCAM, and the pipeline runs in a single forward pass, enabling line-rate processing. Conventional MQTT security baselines such as CPU firewalls, broker ACLs, and cloud IDS are not compared because they operate at different layers and depend on slower software inspection, usually with tens to hundreds of milliseconds of latency. Such comparisons would not meaningfully reflect P4 data-plane behavior.

Future work includes evaluation on hardware P4 targets, integration of in-band telemetry for adaptive policies, and extensions to MQTT-SN and CoAP for broader IoT coverage.

VI. Acknowledgment

This work was supported by JST-ALCA-Next (JPMJAN23F4) and the Vietnam Ambassador.

References

[1] D. Reinsel, J. Gantz, and J. Rydning, "The digitization of the world: From edge to core," IDC White Paper, Sponsored by Seagate Technology, Tech. Rep. US44413318, November 2018, data refreshed May 2020.

[2] S. A. G. Asgar, N. Dong, and D. Mohaisen, "Analysis of misconfigured iot mqtt deployments and a microsegmentation approach," in *Proceedings of the NDSS Workshop on Security of Things (SDIoTSec)*, San Diego, CA, USA, February 2025.

[3] Unit 42, Palo Alto Networks, "2020 unit 42 iot threat report," Palo Alto Networks, Tech. Rep., March 2020.

[4] NIST, "Cve-2024-6786: Mqtt message path traversal allows arbitrary file read," September 2024.

[5] NIST, "Cve-2024-31409: Cyberpower powerpanel business — certain mqtt wildcards not blocked (data exposure risk)," May 2024.

[6] NIST, "Cve-2024-31041: Nanomq 0.21.7 topic filtern null pointer dereference (dos)," April 2024.

[7] B. Asulba, P. F. Souto, and L. Almeida, "Bringing iot intrusion detection to the edge," in *Proceedings of the 8th International Conference on Future Networks and Distributed Systems (ICFNDS '24)*. New York, NY, USA: Association for Computing Machinery, 2025, pp. 295–304.

[8] A. AlSabeh, E. Kfoury, J. Crichigno, and E. Bou-Harb, "P4ddpi: Securing p4-programmable data plane networks via dns deep packet inspection," in *Proceedings of the Workshop on Measurements, Attacks, and Defenses for the Web (MADWeb)*, San Diego, CA, USA, April 2022.

[9] R. Banno and K. Osawa, "Acceleration of mqtt-sn protocol using p4," in *Proceedings of the 2022 IEEE 11th International Conference on Cloud Networking (CloudNet)*, 2022, pp. 16–21.