

Flow-Level SDN Redirection of Detected Attackers to a Digital Twin

Deepraj Singh Makhloga*, Amit Barve*, and Ayan Mondal†

*Military College of Telecommunication & Engineering, Mhow, M.P., India

†Department of Computer Science and Engineering, Indian Institute of Technology Indore, M.P., India

Emails: deepraj236@gmail.com; barve.amit@gmail.com; ayanm@iiti.ac.in

Abstract—Reactive defenses face a trade-off: blocking a detected probe ends observation and may alert the attacker, while allowing it to continue risks production compromise. This paper presents a flow-level redirection mechanism that uses software-defined networking (SDN) to send every *subsequent* connection from a detected attacker to a periodically state-updated digital twin. A Ryu controller installs forward and reverse OpenFlow rules matched on source IP, destination IP, and destination port, rewriting both connection directions without a refused connection or reset. The mechanism changes packet headers only; it does not preserve a connection already open at detection time, which would require TCP-state replication, sequence-number translation, or connection splicing. Measurements of the twin state-synchronization channel over five weeks of live operation (4,003 sampled windows and more than 3.7×10^6 messages) show a median transport latency of 0.634 ms, with median latency below 2 ms in all windows and 95th-percentile latency below 2 ms in 97.5% of windows. The same substrate supports an exploratory catch-all path, an unvalidated anomaly heuristic, and a per-(source-IP, attack-type) fingerprint store that distinguishes first sightings from repeats. Packet captures, including one continuous attacker-side capture, show complete post-redirect connections without retransmissions or resets, while deployment testing exposed three implementation defects with broader architectural implications. A replay demonstration and an eight-source sequential scale sweep exercise classification logging and cumulative flow installation under live traffic.

Index Terms—software-defined networking, OpenFlow, digital twin, cyber deception, intrusion detection

I. INTRODUCTION

High-value servers are attractive targets precisely because they cannot be taken offline to study an intrusion. A defender who blocks a probe the instant it is recognized learns nothing further about the adversary's tooling, and tips the adversary off that the probe was seen; a defender who lets the probe continue against production risks exactly the compromise the detection was meant to prevent. Deception – diverting an attacker onto a decoy that behaves like the real system – has long promised a way out of this trade-off, but classical honeypot placement is typically limited to traffic that can be steered at connection setup, before an attacker has already engaged the real target [1].

This paper presents a redirection substrate implemented in a digital-twin-based cyber-defense testbed for high-value servers and evaluates it against live, reproducible traffic rather than simulation. The mechanism sits at the SDN flow-table layer: a Ryu [2] controller installs a matched forward/reverse

OpenFlow [3] rule pair, keyed on (source IP, destination IP, destination port), that rewrites both legs of every new connection an already-detected attacker opens, so each such connection is served by a periodically state-updated digital twin of the physical host instead of the physical host itself, with no refusal or reset in the evaluated connections. The claim is scoped precisely: the rewrite operates on IP/MAC headers alone, so it applies to connections opened after a redirect installs, not to a connection already open at detection time, which would need TCP-state replication, sequence-number translation, or connection splicing that this system does not implement. Detection is provided by Suricata [4], and the same substrate has an exploratory extension comprising a lightweight anomaly scorer and a persistent per-attacker fingerprint store, to traffic that matches no known attack signature – evaluated here with a single hand-typed payload absent from the test workload, which serves as a proof-of-concept exercise of the mechanism rather than a validated detector.

The contributions of this paper are:

- A concrete SDN flow-rule construction – a forward+reverse NAT-style pair with per-direction IP↔MAC learning – that preserves bidirectional addressing for an attacker's subsequent connections to a live digital twin, explicitly scoped against the stronger, unimplemented claim of mid-connection session migration.
- An architectural finding, surfaced by a real bug in deployment, that redirected attackers' subsequent traffic never reaches the physical host's own vantage point, so detection must be duplicated on the twin's segment.
- An exploratory catch-all and new/replay classification layer on the same substrate, with its scoring/fingerprinting limitations stated explicitly.
- An evaluation from live, logged operation – packet captures, a multi-week transport-latency measurement, and a previously unseen-payload case study, replay logging, and an eight-source sequential scale sweep.

II. RELATED WORK

Honeypots and deception. Classical honeypot design [1], [5] uses fixed decoys or connection-setup redirection. More directly, SDNHive performs controller-based blocking and transparent rerouting to honeypots [6], while SMASH combines SDN, moving-target defense, and managed honeypot

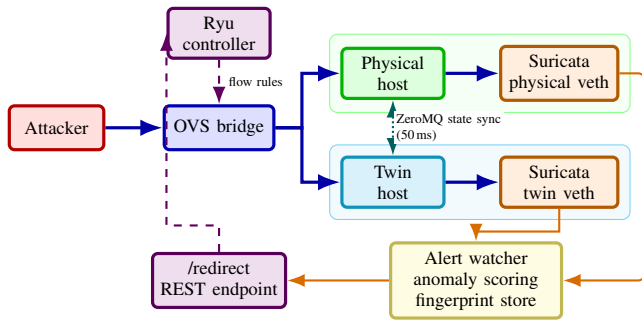


Fig. 1. Experimental architecture: an SDN-controlled bridge carries attacker traffic to either the physical host or its twin; independent Suricata sensors on *both* hosts’ veths feed a watcher that scores, classifies, and (for a new sighting) triggers a Ryu-installed redirect; a ZeroMQ channel keeps the twin’s live state synchronized with the physical host.

deployment [7]. Consequently, traffic redirection itself is not claimed as novel; the contribution is the evaluated bidirectional rule construction for a state-updated twin, together with its measured scope and failure modes.

SDN response and flow migration. SDN’s programmable data plane [3], [8] supports rapid response, but routing packets to another endpoint is distinct from migrating an established transport connection. OpenNF transfers network-function state [9]; a host with no TCP control block instead returns a reset under RFC 9293 [10]. The mechanism here therefore redirects only connections opened after detection.

Digital twins in cyber defense. The digital-twin concept is established in industrial systems [11]. Recent work explicitly identifies the potential intersection of digital twins and cyber deception, while emphasizing realism and restoration challenges [12]. This paper supplies a working network prototype, but does not evaluate semantic twin fidelity at redirect time.

III. SYSTEM OVERVIEW: EXPERIMENTAL TESTBED

The experimental platform is a digital-twin-based cyber-defense testbed built around a physical host and a periodically state-updated twin, both running the same containerized service stack (nginx plus a deliberately vulnerable Flask application used as the attack surface). Fig. 1 shows the pieces this paper is concerned with. A state-synchronization agent streams the physical host’s resource and process state to the twin over ZeroMQ [13] PUB/SUB at 50 ms resolution. The transport delay of this channel is measured over five weeks of live operation. An Open vSwitch (OVS) bridge under Ryu control connects an attacker host, the physical host, and the twin host on one logical segment. Suricata instances tap both the physical host’s and the twin’s OVS ports independently; deployment testing showed that both sensing points are structurally necessary. A watcher process tails each Suricata instance’s alert stream, classifies each alert, and either installs a redirect via Ryu’s REST API or hands the alert to a separately trained reinforcement-learning response agent for a blocking action – the latter is outside this paper’s scope.

IV. THREAT MODEL AND DESIGN GOALS

The threat model assumes an attacker with network reachability to the protected host’s exposed service ports, capable of both scripted and hand-crafted requests, who does not control the SDN control plane or the hypervisor/host running the twin. The attacker is not assumed to be signature-matched in advance; the exploratory catch-all scorer is intended to exercise traffic not covered by known signatures. The system does not defend against an attacker who has already achieved code execution on the physical host prior to detection, and it does not claim to alter the connection that triggered detection itself; the mechanism’s job is to prevent further, new attacker-controlled connections from reaching production once a first request trips detection, not to remediate an existing compromise or migrate an in-progress session.

These operational requirements motivate three design goals:

- 1) **Connection-level redirect transparency.** Once installed, a redirect must give an attacker’s next connection attempt no address mismatch, refused connection, or reset in the evaluated path. This is a claim about new connections, not application-level indistinguishability or continuity of a connection already open at detection time.
- 2) **Production isolation.** From the moment a redirect is installed, no further attacker-originated connection may reach the physical host’s application logic.
- 3) **Continued observability.** The defender must still be able to observe the attacker’s subsequent behavior post-redirect, which is the entire point of choosing redirection over an outright block.

V. FLOW-LEVEL REDIRECTION MECHANISM

A. Detection trigger

The primary Suricata rule matches unauthenticated access to the protected application’s administrative path. Two marker signatures cover the vulnerable API endpoints, while catch-all rules on those endpoints feed the exploratory anomaly-scoring path. On a match, a watcher tails Suricata’s `eve.json` output via filesystem-change notification, extracts a (source IP, destination IP, destination port) triple from the alert record – deliberately not the full 5-tuple, since source port is omitted as a granularity and flow-table-complexity trade-off – and issues a single REST call to the Ryu controller’s `/redirect` endpoint.

B. Flow-rule construction

The Ryu application extends a base L2 learning switch with per-packet IP↔MAC learning, so that the triple from a Suricata alert can be turned into concrete destination-MAC and output-port actions. On a redirect request it installs two flow rules, both at a higher priority (100) than the base learning-switch flows (priorities 0–1), so a redirect always wins over ordinary L2 forwarding for the matched triple – and, because source port is not part of the match, for *every* new connection the attacker opens to that host and port thereafter, not only a

TABLE I
FORWARD AND REVERSE OPENFLOW RULES THAT REDIRECT AN
ATTACKER'S SUBSEQUENT CONNECTIONS WITH BIDIRECTIONAL ADDRESS
CONSISTENCY.

Flow direction	Match fields	Actions
Forward	ip_proto=TCP, ipv4_src=attacker, ipv4_dst=physical, tcp_dst=port	Rewrite ipv4_dst→twin; rewrite eth_dst→twin's MAC; output to twin's port.
Reverse	in_port=twin's port, ipv4_src=twin, ipv4_dst=attacker, tcp_src=port	Rewrite ipv4_src→physical; rewrite eth_src→physical's MAC; output to attacker's port.

single flagged one. Table I gives the exact match/action pair for each direction.

The reverse rule is what makes each subsequent connection's addressing consistent: without it, the twin's replies would carry its own source address and the attacker's stack would drop them as belonging to a connection it never opened. With it, a new connection opened after the redirect installs is addressed, on both legs, exactly as if the physical host answered it. **This says nothing about a connection already open when the redirect installs** – it retains whatever TCP control block the physical host allocated, and redirecting its later packets to the twin without also moving that state would draw a reset from a host that never performed the handshake (RFC 9293 [10]). This system implements neither sequence-number translation, TCP-state replication, nor connection splicing, so it claims nothing about connections already in progress at detection time. The mechanism's actual scope – transparency for new post-detection connections – is verified with physical-, twin-, and attacker-side packet captures.

VI. EXPLORATORY CATCH-ALL SCORING AND NEW/REPLAY CLASSIFICATION

Signature rules only match literal marker strings planted by two test payload generators, and can by definition miss other payloads. As an exploratory extension, two broad catch-all rules fire on any traffic reaching the protected application's deliberately vulnerable endpoints, known attack or not. Because no legitimate traffic in this closed testbed should ever reach those endpoints, a hit there is already a real signal; a lightweight anomaly scorer grades how anomalous a given hit looks, using only fields already present in Suricata's `eve.json` alert record (no payload capture is needed):

$$\text{score} = 0.5 \cdot \text{clamp}_{[0,1]} \left(\frac{b - 150}{450} \right) + 0.5 \cdot \text{clamp}_{[0,1]} \left(\frac{r}{5} \right) \quad (1)$$

where b is request size in bytes and r is the count of alerts from the same source IP in the trailing 60 s window; a request is flagged anomalous at score ≥ 0.3 . This heuristic and its threshold are hand-chosen, not fit or validated against a labeled corpus or benign traffic, and no precision/recall or false-positive-rate figures are reported. The scorer is therefore not presented as a validated detector.

Every alert – signature-matched or catch-all – is then classified by a persistent fingerprint store keyed on (attacker_ip, attack_type): *new* if this exact pair has not been recorded before, *replay* otherwise. A new sighting against the physical host triggers the bidirectional flow-rule redirect on that attacker's subsequent connections, observing them on the twin; a replay does not install another redirect and instead hands the alert to a separately trained response agent, which applies a blocking action to subsequent traffic. Because detection follows packet arrival, the triggering replay can already have reached a previously installed twin path. This IP-plus-type key is a simplification with real failure modes rather than a robust attacker identity: it conflates clients behind NAT, permits spoofing-based evasion, and ignores payload severity.

A. Case study: an unseen test payload

To exercise the catch-all/anomaly-scoring code path beyond the two pre-canned marker signatures, a single request was hand-typed against `/api/echo` that matched neither sid 1000002 nor any string literal anywhere in the attack scripts. It was caught by the sid 1000004 catch-all, scored at anomaly 0.39 (above the 0.3 threshold) by Eq. (1), classified *new* on first contact – triggering a redirect of that attacker's subsequent connections and observation on the twin – and classified *replay* on a second, identical contact, which was blocked with a real `iptables` rule rather than observed again. This is reported as a single, manually-constructed proof that the code path works end to end, not as a validation of detection accuracy against novel attacks in general; that would require evaluation against a labeled corpus of benign and malicious traffic.

VII. EVALUATION

All reported results come from live operation of the experimental testbed and were logged to SQLite/JSONL stores; none are simulated.

A. Connection-level redirect transparency: packet-capture evidence

Table II summarizes two independent packet captures taken during live verification: one on the physical host's veth before the redirect and one on the twin's veth afterward. These are separate connections, not a single migrated flow. Each capture contains ten packets covering the TCP handshake, HTTP request/response exchange, and FIN teardown, with no retransmissions or resets. The captures therefore show that a connection opened after rule installation can complete normally, with no visible refusal or reset. They do not show that an already-open connection can survive redirection; that case was not exercised. As Table I shows and Fig. 1 depicts, the attacker's own socket in the post-redirect case is still addressed to the physical host's IP; the twin-side destination address in Table II reflects what the packet looks like after the OVS rewrite, on the segment beyond it, exactly as designed.

A separate continuous capture on the attacker's interface covered two ordinary HTTP exchanges using different source ports (48518 and 56510). Both completed with handshake,

TABLE II
PACKET-CAPTURE SUMMARY FOR SEPARATE PRE- AND POST-REDIRECT
TCP CONNECTIONS (10 PACKETS EACH).

Measure	Before redirect (physical veth)	After redirect (twin veth)
Observed destination	10.10.0.1:80	10.10.0.2:80
Connection sequence	Handshake; request/response; FIN teardown	Handshake; request/response; FIN teardown
Retransmissions	0	0
TCP resets	0	0
Packets captured	10	10

request/response, and FIN teardown, with no reset or retransmission; the evaluation database independently recorded both as twin-served. Together, these captures show address consistency and normal TCP completion for the observed connections, not general equivalence of application content, timing, or service fingerprints.

B. Twin state-synchronization transport latency

The redirect substrate’s usefulness depends on the twin’s state actually being current when an attacker’s subsequent connections are diverted to it, which is influenced by (but not the same thing as) the transport delay of the ZeroMQ state-sync channel. The evaluation sampled 4,003 logged latency windows (one window per 1,000 streamed state updates) spread across the full multi-week deployment period, plus the most recent 2,000 windows in full detail ($> 3.7 \times 10^6$ records total). What this measures and does not: wall-clock ZeroMQ delivery latency between the sync agent and the twin – not TCP-state synchronization (there is none), application-state correctness, redirect-installation latency, or end-to-end staleness, the last of which must add the 50ms sampling interval, giving a worst-case bound of roughly measured latency plus 50 ms, not latency alone. The one-way calculation also assumes a common or synchronized sender/receiver clock; residual clock error was not measured separately.

Fig. 2 (top) plots every sampled window’s median and 95th-percentile transport latency on a log scale across the whole deployment period. Median latency stayed under 2 ms in 100% of the 4,003 sampled windows, with a deployment-wide median-of-medians of 0.634 ms. The 95th percentile stayed under 2 ms in 97.5% of windows; a small number of windows (33 of 4,003, 0.8%) show a multi-second p95, which were traced to session-boundary events (container restarts between development sessions) rather than steady-state behavior under load. Fig. 2 (bottom) restricts to the most recent 2,000 logged windows – steady-state operation away from any restart boundary – where median latency is 0.776 ms and the 95th percentile stays a tight 1.2–1.3 ms throughout.

C. Replay behavior and sequential scale sweep

A repeated-contact experiment increased the attack-log count from 4 to 6 and the independently verified twin-served count from 2 to 4 (Fig. 3). The corresponding records were

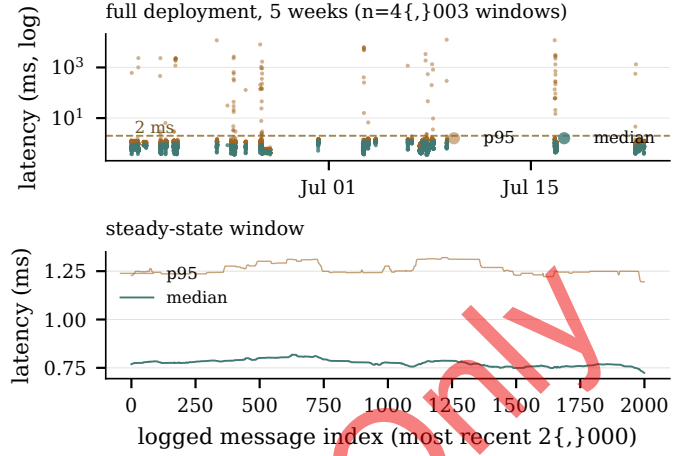


Fig. 2. Twin state-sync channel transport latency. Top: every sampled window ($n = 4,003$) across five weeks of live deployment, log scale – median (teal) never crosses the 2 ms line; a small number of p95 (tan) outliers reach the multi-second range at session boundaries. Bottom: the most recent 2,000 logged windows, showing steady-state behavior away from any restart boundary.

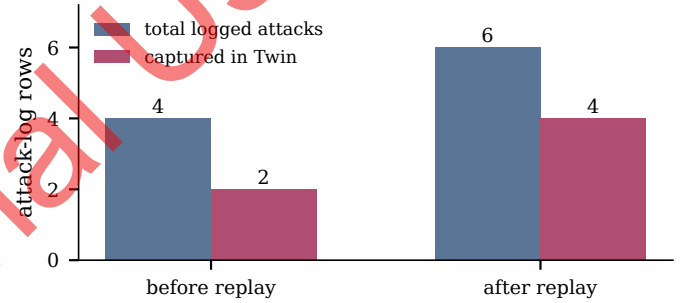


Fig. 3. Cumulative counts before and after the replay demonstration. The plot shows logging and serving location; replay labels were obtained from the corresponding event records.

labeled as replays and forwarded to the blocking response. The twin-served count can still increase because the triggering packet reaches an existing redirect path before the alert-driven block takes effect. Thus, the figure validates cumulative logging and placement, not classification accuracy by itself.

Across ten logged redirect installations, latency ranged from 3.61 to 7.60 ms (median 5.86 ms). The scale sweep then redirected eight IP aliases on one attacker host in sequence (Fig. 4); the OVS table grew from 13 to 27 entries, exactly two rules per redirect. An interleaved benign control connection showed no systematic latency degradation. This measures cumulative rule-table and control-plane cost, not simultaneous multi-host concurrency.

VIII. LESSONS FROM LIVE DEPLOYMENT

Table III summarizes three defects revealed while bringing the mechanism up against a real OVS/Ryu/Suricata stack. Unit tests did not catch them; each required the live network path to be exercised. Two provide generalizable findings rather than merely implementation-specific fixes.

TABLE III
REAL DEFECTS FOUND AND FIXED DURING LIVE DEPLOYMENT OF THE REDIRECTION SUBSTRATE.

Defect	Root cause	Fix
ARP/IP learning race	An ARP-learned flow entry for a (port, dst-MAC) pair also satisfied that direction's first IP packet, so the controller never saw it to learn the host's IP $\leftrightarrow$ MAC binding.	Add <code>eth_type</code> to the learned-flow match so ARP and IP traffic are learned on separate table-miss passes.
Physical-side blind spot	Once an attacker is redirected, all of their subsequent traffic to the (spoofed) physical address is actually delivered to the twin – the physical host's own veth goes structurally silent for that attacker, so a single Suricata instance there cannot see post-redirect activity.	Run a second, independent Suricata instance on the twin's veth (the "twin-side logger").
Self-referential flow corruption	The watcher tails both veths; on the twin's veth, alerts fire on traffic already redirected, with <code>dest_ip</code> already rewritten to the twin's own address. Installing a redirect unconditionally in that case corrupted the OVS flow table with a rule pointing the twin at itself.	Only install a redirect when the alert's <code>dest_ip</code> is genuinely the physical host's address.

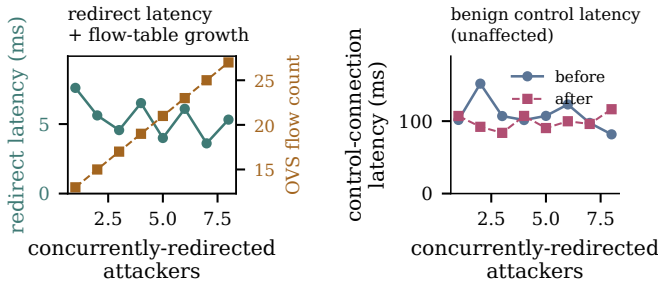


Fig. 4. Eight-source sequential scale sweep using same-host IP aliases. Redirect latency remains stable as the OVS table grows; the benign control connection shows no systematic degradation.

Twin-side logging is structural, not incidental. A honeypot with no physical counterpart has no reason to surface the physical-side blind spot – watching the decoy is the only option anyway. A digital-twin deception mechanism specifically inherits an extra requirement: detection must be provisioned on *both* sides of the redirect, because the physical side's own sensor goes dark for exactly the traffic the defender most wants to keep watching.

Ordering-dependent deduplication is a documented simplification. A marker-laden request matches both its specific signature and the broader catch-all rule; the watcher deduplicates by the Suricata flow identifier and keeps the first alert seen, relying on Suricata evaluating the specific rule before the catch-all for a given rule-file load order – true in this deployment but not a guaranteed property of Suricata's evaluation order in general, and recorded here rather than silently depended upon.

IX. DISCUSSION AND LIMITATIONS

Transport scope. The mechanism rewrites IP/MAC headers only. An already-open connection retains TCP state solely at the physical host, so redirecting its later packets would cause the twin to return a reset under RFC 9293 [10]. Mid-session preservation would require state replication, sequence translation, connection splicing, or socket migration, none of which is implemented.

Evidence boundaries. The captures establish address consistency and normal TCP completion *only* for the observed connections; they do not test response-content equivalence, service fingerprinting, or attacker-side latency distributions. Production isolation follows from the installed rules and twin-side observations but was not verified with simultaneous counters on every physical interface. Likewise, the synchronization experiment measures message transport rather than semantic state fidelity or end-to-end freshness. The scale sweep uses eight sequential IP aliases on one host, not simultaneous independent attackers. No proxy/static-honeypot latency baseline or hardware/software-version sensitivity study was collected, so the timing results are deployment-specific rather than comparative.

Exploratory classification. Eq. (1) is hand-chosen and has no reported precision, recall, or false-positive rate. The fingerprint key (source IP, attack type) conflates clients behind NAT, permits spoofing-based evasion, and ignores payload severity. The case study demonstrates control flow, not detector accuracy; production use requires a labeled benign/attack corpus and a stronger identity or payload fingerprint.

Operational boundaries. Redirects match (source IP, destination IP, destination port), so every later connection from that source to the protected port is diverted. Path-aware steering would require finer rules or application-layer inspection. The SDN control plane is trusted; deployment must protect the Ryu API and OVS management plane through authentication and network isolation.

X. CONCLUSION AND FUTURE WORK

This paper presented bidirectional SDN rules that redirect a detected attacker's subsequent connections to a periodically state-updated digital twin. It does not preserve an already-open TCP connection. Evidence from a live OVS/Ryu/Suricata stack includes physical-, twin-, and attacker-side captures, a five-week 4,003-window transport-delay measurement, replay logging, an eight-source sequential scale sweep, and three deployment defects. These results support connection-level redirection and implementation feasibility, not general application indistinguishability, validated anomaly detection, or production-scale concurrency.

Future work includes genuine mid-session preservation via state replication, sequence-number translation, or connection splicing; path-aware redirect granularity; a specificity-ranked alternative to first-seen-wins deduplication; a trained, precision/recall-validated anomaly detector; a more robust replay-classifier key than source IP plus attack type; and evaluation with larger attacker and benign traffic populations.

REFERENCES

- [1] N. Provos, "A virtual honeypot framework," in *Proc. 13th USENIX Security Symp.*, San Diego, CA, USA, 2004, pp. 1–14.
- [2] Ryu SDN Framework Community, "Ryu SDN framework." [Online]. Available: <https://ryu-sdn.org>
- [3] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner, "OpenFlow: Enabling innovation in campus networks," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 38, no. 2, pp. 69–74, 2008.
- [4] Open Information Security Foundation (OISF), "Suricata: Network threat detection engine." [Online]. Available: <https://suricata.io>
- [5] L. Spitzner, *Honeypots: Tracking Hackers*. Boston, MA, USA: Addison-Wesley, 2002.
- [6] M. Karakate, H. Esaki, and H. Ochiai, "SDNHive: A proof-of-concept SDN and honeypot system for defending against internal threats," in *Proc. 11th Int. Conf. Communication and Network Security (ICCNS)*, 2021, pp. 9–20.
- [7] N. d'Ambrosio, C. Lista, G. Perrone, and S. P. Romano, "SMASH: An SDN-MTD framework for efficient honeypot deployment and insider threat mitigation," *Comput. Netw.*, vol. 269, Art. no. 111327, 2025.
- [8] D. Kreutz, F. M. V. Ramos, P. E. Veríssimo, C. E. Rothenberg, S. Azodolmolky, and S. Uhlig, "Software-defined networking: A comprehensive survey," *Proc. IEEE*, vol. 103, no. 1, pp. 14–76, Jan. 2015, doi: 10.1109/JPROC.2014.2371999.
- [9] A. Gember-Jacobson et al., "OpenNF: Enabling innovation in network function control," in *Proc. ACM SIGCOMM*, Chicago, IL, USA, 2014.
- [10] W. Eddy, Ed., "Transmission Control Protocol (TCP)," IETF RFC 9293, Aug. 2022.
- [11] F. Tao, H. Zhang, A. Liu, and A. Y. C. Nee, "Digital twin in industry: State-of-the-art," *IEEE Trans. Ind. Informat.*, vol. 15, no. 4, pp. 2405–2415, Apr. 2019.
- [12] J. Heluany, A. Amro, V. Gkioulos, and S. Katsikas, "Interplay of digital twins and cyber deception: Unraveling paths for technological advancements," in *Proc. ACM/IEEE EnCyCriS/SVM*, Lisbon, Portugal, 2024, pp. 20–28.
- [13] P. Hintjens, *ZeroMQ: Messaging for Many Applications*. Sebastopol, CA, USA: O'Reilly Media, 2013.