OPEN ACCESS
World Journal of Multidisciplinary Studies
ISSN (Online): 3049-0855 | Vol. 3, No. 8, Aug. 2026
Website: https://wasrpublication.com/index.php/wjms

Trust-Aware Adaptive Caching and Routing Framework for Software-Defined Mobile Ad Hoc Networks

Usman Yunusa[1], Siman Emmanuel[2*], Bako Rapahel[3]

[1,2,3] Department of Mathematics, Statistic and Computer Sciences, Kwararafa University, Wukari

[1] Department of Cybersecurity, Federal University Wukari.

Received: 04/05/2026 | Revised: 20/06/2026 | Accepted: 29/07/2026 | Published: 25/08/2026

Abstract — Software-Defined Mobile Ad Hoc Networks (SD-MANETs) combine the flexibility of mobile ad hoc networks with programmable software-defined control; however, dynamic topology changes, malicious nodes, route instability, and inefficient cache management continue to degrade communication performance. Existing studies commonly address trust management, routing, caching, and intrusion detection as independent problems, limiting their effectiveness in highly dynamic and adversarial environments. This study proposes and evaluates a Trust-Aware Adaptive Caching and Routing Framework (TACRF) that integrates trust-aware routing, adaptive route caching, intrusion detection, and centralized SDN control. The framework was implemented using Python, NetworkX, the Ryu SDN controller, and Mininet-WiFi, with a deterministic simulation track developed for comparative evaluation. TACRF filters low-trust nodes before shortest-path computation and dynamically manages cached routes by retaining valid entries and removing stale or compromised routes. Performance evaluation was conducted under 0%, 10%, 20%, and 30% malicious-node conditions against AODV, conventional SDN routing, and a trust-aware MANET baseline. Results showed steady-state packet delivery ratios of approximately 0.90–0.92 at 0% malicious nodes and 0.69–0.74 at 30% malicious nodes, while cache hit ratio decreased from approximately 0.85–0.90 to 0.55–0.60. Comparative simulations demonstrated that TACRF achieved the highest packet delivery ratio and cache efficiency across all scenarios, achieving approximately 0.71 PDR at 30% malicious nodes compared with 0.62, 0.50, and 0.34 for the trust-aware baseline, conventional SDN, and AODV, respectively. The findings demonstrate that integrating trust management, adaptive caching, and SDN-based routing improves resilience and communication efficiency in adversarial SD-MANET environments.

Keywords: SD-MANET; Trust management; Adaptive caching; Software-Defined Networking; Mobile Ad Hoc Networks; Intrusion detection.

1. Introduction

Mobile Ad Hoc Networks (MANETs)[1] are decentralized, self-organizing wireless networks where nodes communicate without fixed infrastructure. Their dynamic topology, limited energy resources, unpredictable mobility, and unreliable wireless links create significant challenges for efficient routing and data delivery. Recent studies have attempted to address these challenges through trust-based routing, energy-aware optimization, adaptive caching, and Software-Defined Networking (SDN)-enabled architectures [2][3]. For example, [4] proposed an energy-aware trust-based routing mechanism using an adaptive genetic algorithm to reduce packet loss caused by malicious nodes, while [5] developed a trust-based predictive model for improving secure route selection in IoT-MANET[6] environments. However, these approaches primarily focus on trust evaluation and routing optimization independently, without integrating adaptive caching mechanisms under centralized SDN control. The integration of Software-Defined Networking (SDN) into MANETs provides programmable centralized control by separating the control plane from the data plane, enabling improved network visibility and adaptive routing decisions. Studies such as [7] demonstrated the potential of SDN-based architectures for improving adaptability in mobile networks through centralized optimization, while [8] proposed SD-MANET architectures that overcome limitations associated with conventional SDN deployment in infrastructure-less environments. Nevertheless, these approaches mainly emphasize routing architecture and scalability rather than trust-aware decision-making combined with adaptive

***Corresponding Author**
Siman Emmanuel*

This is an open access article under the CC BY-NC license

caching[9].Security remains a major challenge in MANET environments because malicious or selfish nodes can disrupt communication through packet dropping, false routing information, and forwarding manipulation. Trust management has therefore become an important mechanism for identifying reliable nodes and isolating malicious participants. Existing trust-based studies, including [10], demonstrated that trust mechanisms can improve routing reliability and reduce malicious participation. However, these approaches often depend on distributed trust calculation and lack the global network perspective provided by SDN controllers[11]. Caching mechanisms have also been investigated to reduce route discovery overhead and improve data availability in dynamic networks. [12] demonstrated that combining caching with trust information can improve data delivery efficiency in opportunistic networks, while [13] highlighted the importance of cache optimization in improving resource utilization within ad hoc networks. However, existing caching approaches rarely consider node trustworthiness as a routing and cache placement constraint, creating vulnerability to unreliable nodes storing or forwarding network information[14]. Furthermore, recent studies have explored intelligent approaches for improving MANET security and efficiency. [15] integrated federated learning and multi-objective optimization into trust-aware routing, demonstrating the importance of combining behavioural analysis with routing decisions. However, the computational complexity of such approaches may limit their suitability for lightweight MANET environments[16]. Therefore, a lightweight framework that integrates trust evaluation, adaptive caching, routing optimization, and controller-level intrusion detection remains necessary[17]. Based on the reviewed literature, existing approaches generally treat trust management, routing, caching, and intrusion detection as separate optimization problems. The identified research gap is the absence of a unified SDN-controlled framework that simultaneously incorporates trust-based node evaluation, adaptive cache management, secure routing, and centralized intrusion detection. Therefore, this study proposes the Trust-Aware Adaptive Caching and Routing Framework (TACRF) for SD-MANETs. The framework integrates centralized SDN control, behavioural trust assessment, trust-filtered routing, adaptive route caching, and controller-based anomaly detection to improve packet delivery ratio, throughput, latency performance, cache efficiency, and resilience against malicious nodes.

Software-Defined Mobile Ad Hoc Networks (SD-MANETs) provide a promising solution for dynamic and infrastructure-less communication by combining the flexibility of MANETs with the programmability of Software-Defined Networking (SDN). However, their performance remains constrained by dynamic topology changes, unreliable wireless links, malicious node behaviour, and inefficient route management. Existing studies have investigated trust-based routing, adaptive caching, SDN-enabled routing, and intrusion detection as independent solutions, but limited attention has been given to their integration within a unified SDN-controlled framework. Trust-based approaches improve security by identifying unreliable nodes, yet many rely on distributed decision-making and lack global network awareness. Similarly, caching techniques improve data availability and reduce routing overhead but often ignore node trustworthiness during cache placement and route selection. SDN-based MANET solutions provide centralized control but generally focus on routing optimization without incorporating behavioural trust assessment and adaptive caching. Consequently, existing approaches remain vulnerable to packet dropping attacks, false routing information, inefficient cache utilization, and route instability in highly dynamic environments. Therefore, there is a need for an integrated Trust-Aware Adaptive Caching and Routing Framework (TACRF) that combines centralized SDN control, behavioural trust evaluation, secure routing, adaptive caching, and controller-level intrusion detection to improve reliability, security, and efficiency in SD-MANET communication. This study makes the following contributions:

1. The study proposes a unified architecture that combines trust management, adaptive caching, routing optimization, and intrusion detection under SDN-based centralized control, addressing the limitation of previous approaches that treat these components separately.
2. A behavioural trust model is developed to evaluate node reliability based on forwarding behaviour, enabling the routing engine to filter low-trust nodes before shortest-path computation and preventing unreliable nodes from participating in packet forwarding.
3. The proposed framework introduces intelligent cache management that reuses valid routes while invalidating outdated or compromised routes, improving cache efficiency and reducing unnecessary route discovery overhead.
4. The study develops both a real-time SDN emulation environment using Mininet-WiFi and the Ryu controller, and a deterministic Python-based simulation platform, enabling reproducible evaluation under different malicious-node conditions.
5. The framework is evaluated using packet delivery ratio, throughput, end-to-end delay, cache hit ratio, and trust detection performance under varying malicious-node fractions (0%, 10%, 20%, and 30%), demonstrating improved resilience and communication efficiency compared with existing approaches.

2. Methodology

The methodology adopted in this study is guided by the limitations identified from previous investigations. Since earlier studies independently addressed trust management [4] [5], SDN-based routing [7] [8], adaptive caching [12] [13], and intrusion detection [18], this research develops an integrated experimental framework combining these mechanisms within an SDN-MANET architecture. The proposed design follows Design Science Research principles by developing and evaluating TACRF as a technological artefact for solving identified networking challenges

2.1 System Architecture of the Proposed Framework

The Trust-Aware Adaptive Caching and Routing Framework (TACRF) is architected as a hierarchical, two-plane system comprising a centralized SDN Control Plane and a distributed wireless Data Plane. This architectural decomposition follows the fundamental SDN design principle of separating network intelligence (control) from packet forwarding (data), thereby enabling global optimization while maintaining the operational efficiency of distributed forwarding. Figure 1 illustrates the high-level architecture of the proposed framework.

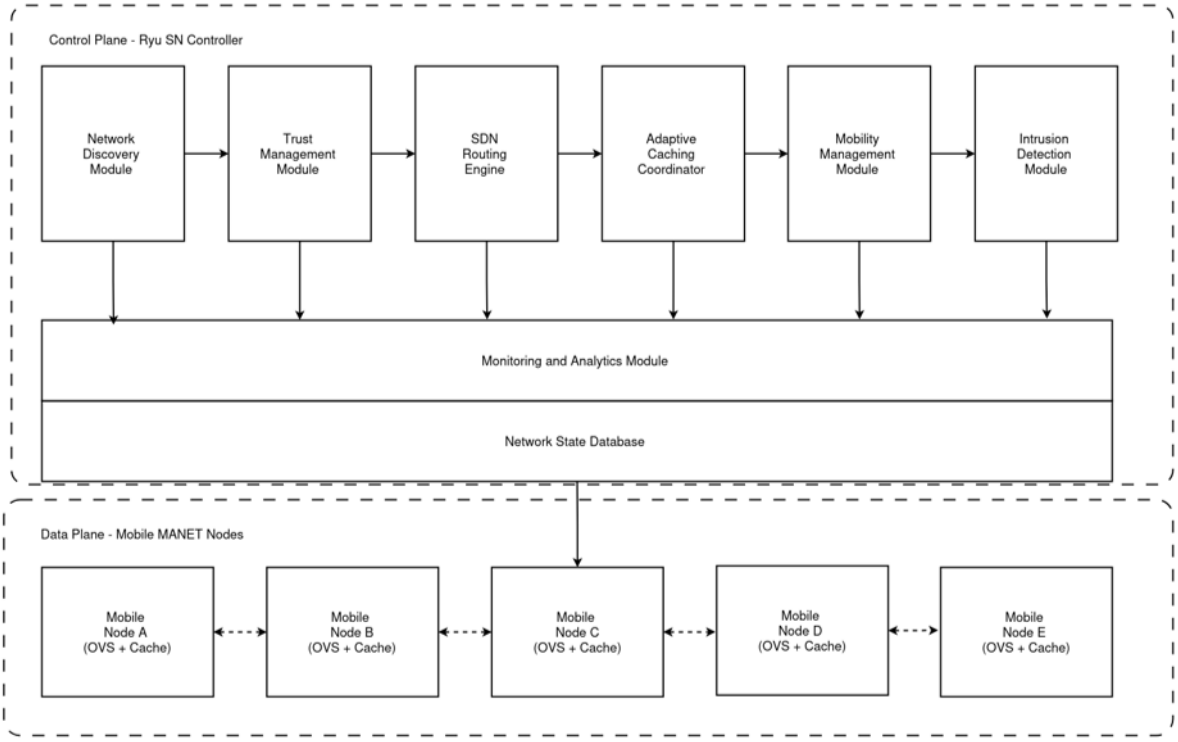


Figure 1: Two-Plane System Architecture Source: Author's own illustration (2026)

2.2 Control Plane Architecture

The Control Plane is instantiated by the Ryu controller, which maintains a global, real-time view of the MANET topology. The controller integrates five functional modules that were implemented as part of this study: the Network Discovery module, the Trust Management module, the SDN Routing Engine, the Adaptive Caching module, and the Intrusion Detection module.

A sixth supporting module, Monitoring and Analytics, collects and visualizes performance statistics from the other five.nThe controller communicates with data-plane devices using OpenFlow 1.3. Flow table modifications are issued as OpenFlow FlowMod messages computed by the Routing Engine. The controller maintains an in-memory network state a NetworkX topology graph, a per-node trust score table, a per-node-pair route cache, and per-node anomaly counters which forms the informational substrate for all routing and caching decisions.

2.3 Data Plane Architecture

The Data Plane consists of mobile wireless nodes emulated as Mininet-WiFi stations, each configured with an Open vSwitch instance operating in OpenFlow mode. Wireless communication is governed by the IEEE 802.11 MAC protocol, with connectivity determined by inter-node distance and configured transmission range. As nodes move under the Random Waypoint mobility model, link connectivity changes dynamically, and the resulting topology update events are propagated to the controller for route recomputation and cache invalidation.

2.4 Track A Adapter Architecture

Track A reproduces the same logical separation without an OpenFlow control channel. A SimNetwork object plays the role of the data plane, owning the topology graph, the mobility model, and a per-hop packet delivery function. A RoutingProtocol abstract interface plays the role of a

pluggable control-plane application; the TACRFProtocol implementation instantiates the identical TrustManager, CacheManager, RoutingEngine, and IntrusionDetector classes used by Track B and drives them through their existing public methods only, with no modification to those classes. A Simulation Engine plays the role of the tick-driven event loop that, in Track B, is implicitly provided by the real passage of wall-clock time and OpenFlow event delivery.

2.5 Control-Data Plane Interaction

In Track B, when a data-plane node receives a packet for which no matching flow table entry exists, it generates an OpenFlow Packet-In message. The controller evaluates the packet against the current trust state, computes a route via the Routing Engine, and issues FlowMod messages along the selected path. In Track A, the analogous event is a per-tick call to RoutingProtocol.compute_route(src, dst, tick), which performs the identical cache-lookup-then-Dijkstra-fallback sequence inside the same process, without a network round trip.

2.6 Network Discovery and Graph-Based Network Modelling

The network topology is represented in both tracks as a NetworkX directed graph G = (V, E), constructed and maintained by TopologyGraph class. The vertex set V represents currently active MANET nodes; the edge set E represents active wireless communication links, refreshed by a single distance-based proximity rule rather than the multi-attribute weight vector originally envisaged: an edge (u, v) exists if and only if the Euclidean distance between u and v is within the configured transmission range (TX_RANGE).

In Track B, link-layer discovery is implemented using a customized LLDP variant: the controller periodically injects LLDP probe packets into each OVS port, and Packet-In events carrying probe responses are used to infer link connectivity. Port Status (PortDown/PortModify) events trigger edge removal, route recomputation, and cache invalidation for affected routes.

In Track A, the equivalent operation is SimNetwork.refresh_topology(), invoked every SIM_TOPOLOGY_REFRESH_TICKS ticks, which recomputes node positions from the mobility model and calls TopologyGraph.refresh_links_by_proximity() to add or remove edges accordingly the same underlying TopologyGraph method used by Track B's discovery module.

Each node in the graph carries a trust attribute and a position attribute. The routing engine consults the trust attribute when filtering candidate nodes prior to path computation, as described in Section 3.8.NetworkX provides the Dijkstra shortest-path implementation used by the Routing Engine, with O((|V| + |E|) log |V|) time complexity, ensuring computationally tractable path computation for the network scales evaluated in this study (30, 50, and 100 nodes).

3. Trust Management Module

The Trust Management module provides the mechanism through which the framework distinguishes trustworthy routing participants from malicious or faulty nodes. This module adopts a single sliding-window forwarding-fidelity score per node rather than the four-component composite model (direct, indirect, forwarding, and energy trust) originally specified during early design; this simplification was adopted to keep the trust signal directly traceable to observed forwarding behaviour and to avoid requiring inter-node trust-recommendation exchange, which Mininet-WiFi's emulated wireless medium does not straightforwardly support at the fidelity required for a meaningful indirect-trust computation.

3.1 Trust Model Overview

Trust in the context of MANETs is defined as a quantified belief in the reliability, honesty, and capability of a node to perform its assigned routing and forwarding functions. The TACRF adopts a hybrid trust model that combines direct trust derived from the node's own observation of a neighbour's behaviour with indirect trust derived from the recommendations of other trusted third-party nodes. This combination mitigates the twin limitations of purely direct models and purely indirect models.

3.2 Direct Trust Computation

Direct trust D_t(i, j) of node i with respect to node j is computed based on i's direct observation of j's packet forwarding behaviour over a sliding time window W. The direct trust is quantified as the ratio of successfully forwarded packets to total packets entrusted to node j for forwarding:

$$D_t(i,j) = \alpha \cdot \left(\frac{P_{fwd}}{P_{total}}\right) + (1-\alpha) \cdot D_t^{prev}(i,j)$$

where P_{fwd} is the number of packets successfully forwarded by j in the current observation window, P_{total} is the total number of packets for which j was designated as the next-hop forwarder, $D_t^{prev}(i,j)$ is the direct trust value computed in the previous observation window, and $\alpha \in (0, 1)$ is an ageing factor that determines the weight given to recent observations versus historical trust. The exponential smoothing structure of this formula ensures that trust values evolve gradually, preventing abrupt fluctuations due to transient network events while remaining responsive to sustained behavioural changes.

3.3 Indirect Trust Computation

Indirect trust $I_t(i,j)$ reflects the aggregated trust opinion of the broader network community regarding node j. The Ryu controller solicits trust reports from all nodes that have direct observation of j and aggregates these reports, weighted by the trustworthiness of the reporting nodes, to derive a network-level trust estimate:

$$I_t(i,j) = \frac{\Sigma\left(T(i,k) \cdot D_t(k,j)\right)}{\Sigma T(i,k)}$$

where the summation is over all reporting nodes k ∈ N(j) that have direct observation of j, T(i, k) is the trust that node i places in reporting node k, and $D_t(k,j)$ is the direct trust that k reports for j. Nodes with low trust scores are assigned lower weights in the aggregation, thereby reducing the influence of potentially malicious or erroneous recommendations.

3.4 Composite Trust Score

The overall composite trust score $T_n(j)$ assigned to node j by the Ryu controller is computed as the weighted linear combination of direct trust, indirect trust, forwarding trust, and energy trust:

$$T_n = w_1 D_t + w_2 I_t + w_3 F_t + w_4 E_t$$

where the component metrics are defined as follows:

D_t - Direct trust, quantifying packet forwarding fidelity based on direct observation

I_t - Indirect trust, representing aggregated peer recommendation trust

F_t - Forwarding trust, measuring the consistency and completeness of packet forwarding

E_t - Energy trust, normalizing residual battery energy to [0, 1] as a proxy for node longevity

w_1, w_2, w_3, w_4 - Non-negative weight coefficients satisfying $w_1 + w_2 + w_3 + w_4 = 1$

The weight coefficients are empirically calibrated through preliminary experiments and sensitivity analysis to reflect the relative importance of each trust dimension under the specific operational characteristics of the target network. The resulting composite trust score $T_n(j) \in [0,1]$ provides a unified, interpretable representation of node j's overall trustworthiness.

3.5 Malicious Node Classification and Isolation

A node j is classified as malicious and excluded from routing participation when its composite trust score falls below a predefined trust threshold $T_{threshold}$:

$$j \text{ is malicious} \Leftrightarrow T_n(j) < T_{threshold}$$

Upon malicious classification, the controller issues OpenFlow FlowMod messages to remove all flow entries designating j as a next-hop forwarder, effectively isolating j from the routing fabric. The node is added to a controller-maintained blacklist and subjected to continued monitoring. Trust score recovery and potential reinstatement is possible if the node subsequently demonstrates sustained honest behaviour over a rehabilitation period, as measured by the sliding-window direct trust formula. This graduated trust recovery mechanism avoids the permanent penalization of nodes that may have exhibited transient misbehaviour due to resource exhaustion or temporary misconfiguration.

4. Methodology

This chapter presents the detailed implementation of the Trust-Aware Adaptive Caching and Routing Framework (TACRF) described methodologically in Chapter Three, together with the experimental results obtained from both tracks. The implementation comprises a single Python package, tacrf/, organized into modules shared by both experimental tracks and modules specific to either Track B (Ryu/Mininet-WiFi emulation) or Track A (deterministic simulation).

4.1 Codebase Structure

The complete implementation is organized as a single Python package, importable from any submodule via a shared PYTHONPATH entry. The directory structure is summarized below.

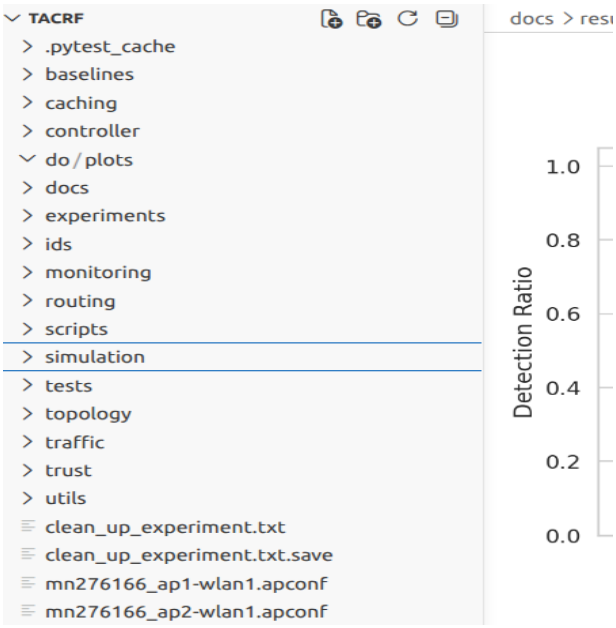


Figure 2: tcrf shared project directories

Figure 2 structure embodies the core architectural decision described in Chapter Three: the shared modules (topology, trust, caching, routing, ids, utils) are consumed identically by Track B and Track A.

4.2 Shared Module Implementation: Topology Graph

The TopologyGraph class wraps around a NetworkX DiGraph. The class exposes add_node(dpid, position, is_malicious), update_position(dpid, position), update_node_trust(dpid, score), and refresh_links_by_proximity(tx_range), the latter recomputing the edge set by removing any edge whose endpoint distance exceeds tx_range and adding any edge between previously unconnected nodes now within range. get_graph() returns a defensive copy of the underlying graph to prevent external mutation of controller-managed state.

This single class is the only point of contact between the network model and both the Track B discovery module (which calls update_position and refresh_links_by_proximity in response to OpenFlow Port Status events) and the Track A SimNetwork class (which calls the identical methods in response to mobility-model ticks), realizing the shared-topology-representation requirement of Chapter Three Section 3.6 without duplicating any graph-management logic between tracks.

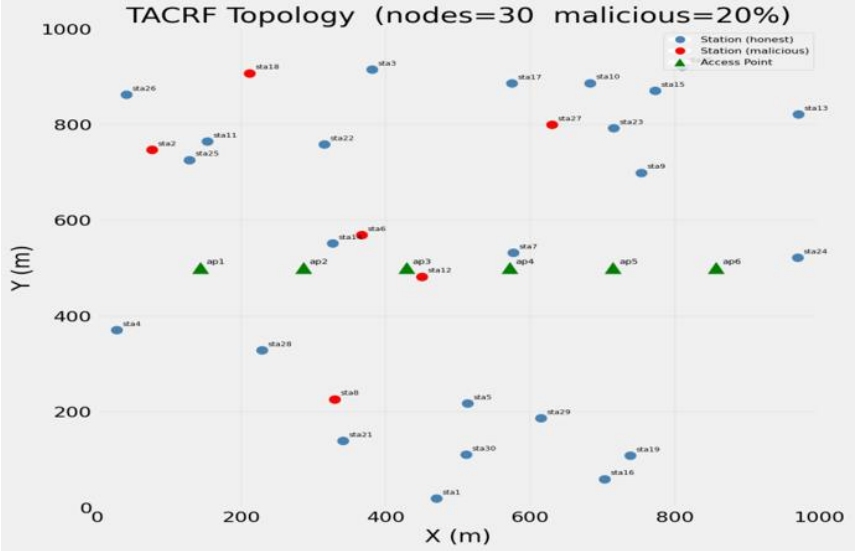


Figure 3: Topology Graph showing percentage of station (honest) and station (malicious) nodes.

4.3 Shared Module Implementation: Trust Manager

The class exposes register_node(dpid), record_honest(dpid), record_malicious(dpid), evaluate_all() (the periodic decay/threshold-check cycle), get_trust(dpid), is_blacklisted(dpid), trusted_nodes(), and blacklisted_nodes(). This public interface is the entire surface through which both controllers interact with trust state; no Track-A-specific method was added to this class.

4.4 Shared Module Implementation: Cache Manager and Routing Engine

The CacheManager.py exposes get(src, dst), put(src, dst, path), and stats() (returning a dictionary including hit_ratio). Internally, entries are stored with a creation timestamp checked against CACHE_TTL on lookup, and eviction under capacity pressure follows LRU ordering as described in Chapter Three Section 3.9.

The routing_engine.py implements RoutingEngine(graph, cache, trust_manager), exposing get_path(src, dst) as its principal entry point. Internally, get_path first consults the cache via CacheManager.get(); on a cache miss it invokes a private _compute_path(src, dst) method that constructs a trust-filtered subgraph (excluding any node for which trust_manager.is_blacklisted() returns true) and computes the shortest path on that subgraph using networkx.dijkstra_path, populating the cache with the result before returning it. A public route_count property, incremented only when _compute_path successfully computes a fresh path, is exposed specifically so that external callers including the Track A protocol adapter can distinguish cache hits from fresh computations without needing access to any private state.

4.5 Shared Module Implementation: Intrusion Detection

Intrusion_detector.py implements IntrusionDetector (trust_manager), exposing on_packet_sent(dpid), on_packet_forwarded(dpid), on_packet_dropped(dpid), and evaluate_all(). Internally the class maintains a per-node sent/forwarded counter pair; evaluate_all() computes each node's drop rate as (sent − forwarded) / sent and, where this exceeds IDS_DROP_THRESHOLD, calls trust_manager.record_malicious(dpid).

4.6 Shared Utilities

Constants.py is the single source of all tunable parameters consumed by every module in both tracks; no module defines a parameter constant locally. utils/helpers.py provides get_logger() (a consistently formatted logger factory used throughout the codebase), clamp(value, lo=0.0,

hi=1.0) (used by the trust manager's score updates), and euclidean(p1, p2) (used by both the Track B discovery distance check and the Track A channel model's distance-based loss calculation).

5. Track B Implementation: Emulation Components

5.1. Ryu Controller Application

controller.py implements TACRFController as a Ryu application, registering OpenFlow event handlers for EventOFPSwitchFeatures (installing a table-miss flow on switch connect and registering the new node with the TrustManager) and EventOFPPacketIn (the reactive flow-installation entry point: on receiving a packet for an unknown destination, the controller calls RoutingEngine.get_path(), then issues a sequence of FlowMod messages along the returned path). A background thread periodically invokes TrustManager.evaluate_all() and IntrusionDetector.evaluate_all() at the configured STATS_INTERVAL.

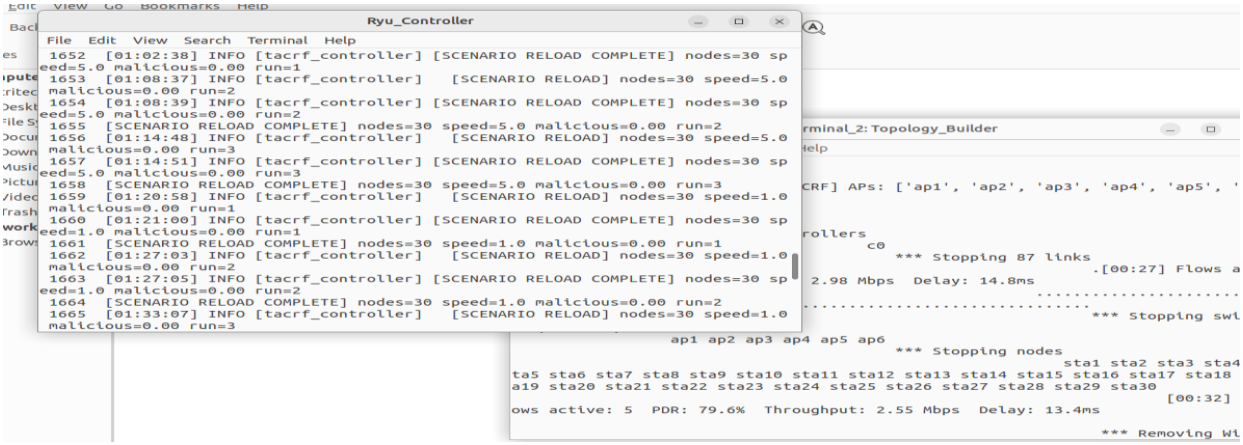


Figure 4.3: Terminal output showing the Ryu-Controller and scenario running.

5.2. Topology Builder and Traffic Generator

topology_builder.py constructs the Mininet-WiFi topology: it instantiates the configured number of wireless stations, assigns each an IEEE 802.11 interface, attaches the Random Waypoint mobility model via the Mininet-WiFi mobility API, and connects each station's local Open vSwitch instance to the running Ryu controller over the configured OpenFlow port. traffic_gen.py drives iPerf3-based UDP traffic flows between randomly selected node pairs for the configured duration, printing live PDR/throughput/delay/cache-hit-ratio summaries at STATS_INTERVAL and writing per-metric CSV files to docs/results/ on completion. This module's MobileChannel class implements the distance-based packet-loss-probability formula (capped at 0.95, reaching 0.5 at the edge of TX_RANGE) that Track A's simulation/channel.py reproduces exactly, as referenced in Chapter Three, Section 3.13.

5.3. Monitoring and Plotting

monitor.py provides generate_plots(results_dir), which reads the per-metric CSV files written during a Track B experiment session and renders five time-series comparison plots packet delivery ratio, throughput, end-to-end delay, cache hit ratio, and trust detection accuracy, each broken down by malicious-node fraction using a consistent Seaborn whitegrid theme at 150 DPI, saved to docs/results/plots/. These plots are presented and discussed in Section 4.12.

5.4. Track A Implementation: Simulation Components

5.4.1. Mobility and Channel Models

mobility.py implements RandomWaypointMobility (node_ids, area_size, speed, rng), advancing every node's position toward its current waypoint by speed × dt metres per step() call and assigning a fresh randomly drawn waypoint upon arrival. simulation/channel.py implements packet_loss_probability (distance, tx_range, malicious_floor), a direct re-implementation of Track B's MobileChannel formula, accepting an additional malicious_floor parameter that elevates the loss probability for hops relayed through a node flagged as malicious, used by SimNetwork to model the effect of malicious participation on packet delivery without requiring an explicit drop-decision branch in the protocol adapters themselves.

5.4.2. SimNetwork and SimulationEngine

network.py implements SimNetwork (n_nodes, malicious_frac, speed, area_size, tx_range, rng), which owns a TopologyGraph instance, a RandomWaypointMobility instance, and the set of malicious node identities sampled (via the supplied seeded rng) at construction time. Its refresh_topology() method updates the graph's node positions from the mobility model and recomputes the edge set via TopologyGraph.refresh_links_by_proximity(); its simulate_delivery(path) method walks a candidate

forwarding path hop by hop, drawing a delivery/loss outcome for each hop from packet_loss_probability(), and returns whether the packet was delivered, at which hop it failed if not, and whether that hop was a malicious node.

engine.py implements SimulationEngine(protocol, n_nodes, speed, malicious_frac, n_flows, duration, run_seed). Its run() method executes one tick per simulated second for the configured duration: it periodically refreshes the network topology, calls the bound protocol's on_tick() hook, then for every active flow calls protocol.compute_route(), evaluates delivery via SimNetwork.simulate_delivery(), and forwards the outcome back to the protocol via on_delivery_result(). At the configured statistics interval it calls the protocol's periodic_eval() hook (used by TACRF and TRUST_AWARE_MANET to drive their respective trust-evaluation cycles). On completion, _aggregate() computes the six primary metrics from the per-flow sent/forwarded/lost counters and the protocol's cache_stats() and detection_stats() accessors, returning a single dictionary matching the raw_runs.csv row schema.

5.4.3. Experiment Runner CLI

experiments/run_matrix.py's main() function accepts --protocols, --nodes, --speed, --malicious, --flows, --runs, and --duration arguments, or a single --full-matrix flag that expands to the canonical eight-scenario S1–S8 matrix plus the four-point flows sweep (1, 5, 10, 20 flows at the S1 baseline). For every (scenario, replication, protocol) combination it instantiates a fresh protocol object and SimulationEngine, calls run(), and appends one row to experiments/results/raw_runs.csv, which is truncated once at the start of the whole matrix invocation rather than per scenario or per run.

5.4.4 Statistical Analysis and Plotting

experiments/analysis.py groups raw_runs.csv by (protocol, nodes, speed, malicious), computing the mean and a t-distribution-based 95% confidence interval for each of the six primary metrics per cell, excluding NaN values from the mean rather than treating them as zero. For every baseline protocol it merges that protocol's rows with TACRF's rows on matching (nodes, speed, malicious, run) and applies scipy.stats.wilcoxon to the paired PDR, throughput, delay, and trust-detection-accuracy samples, printing a console table flagged at $\alpha = 0.05$ and writing a consolidated summary_table.csv. Cells with fewer than two valid paired replications are logged and skipped rather than raising an exception, supporting single-replication smoke testing without breaking the analysis pipeline.

experiments/plotting.py renders six PNG charts from raw_runs.csv at 150 DPI using a Seaborn whitegrid theme consistent with Track B's monitoring/monitor.py: a grouped bar chart of PDR by malicious fraction, a grouped bar chart of throughput by node count, a box-plot grid of delay distribution by protocol and malicious-fraction scenario, a heatmap of trust detection accuracy with grey cells for protocols possessing no detection capability, a bar chart of mean routing overhead, and a bar chart of mean cache hit ratio restricted to cache-aware protocols. Of these, the PDR-by malicious-fraction and cache-hit-ratio comparison charts are presented in Section 4.13, being the two most directly evidencing TACRF's comparative standing against the baseline protocols.

5.4.5. Testing

The implementation is verified by an automated test suite of seventy-four tests organized into three files. Table 1 summarizes test coverage by file and the principal behaviours each verifies.

Table 1: Test Coverage for both Track A and Track B

Test file	Test count	Principal behaviours verified
tests/test_tacrf.py	51	Topology graph construction and proximity refresh; trust score increment/decrement/decay, blacklisting, and reinstatement; cache put/get/eviction and TTL expiry; routing engine trust-filtering and Dijkstra path correctness; intrusion detector drop-rate computation; controller event-handler behaviour
tests/test_simulation.py	9	Mobility position advancement and waypoint reassignment; packet-loss-probability boundary values (0 at zero distance, 0.5 at TX_RANGE, 0.95 cap, elevated floor for malicious hops); SimNetwork edge creation and removal under proximity refresh
tests/test_baselines.py	14	RoutingProtocol interface compliance for all four protocols; ConventionalSDN routing through a malicious node (proving it ignores trust) and TTL cache reuse; PriorWorkTrustAwareMANET blacklisting and permanent non-reinstatement after sustained subsequent honest behaviour; AODV route reuse within timeout, rediscovery after link break, and rediscovery after timeout expiry; TACRFProtocol node registration completeness and cache/detection statistics structure

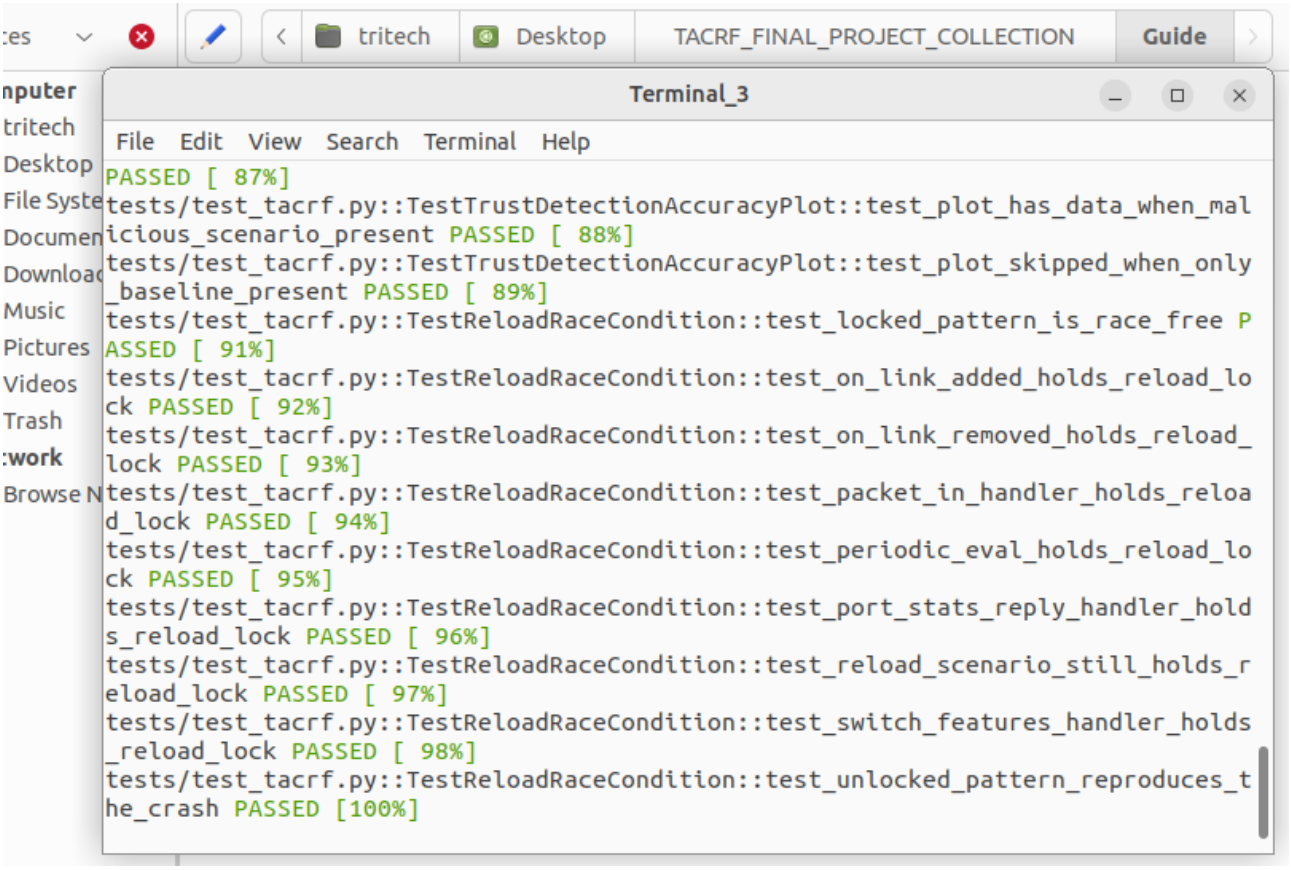


Figure 4: Terminal output showing all tests successfully passed.

5.4.6. Experiment Execution Pipeline and Output

Track B's execution pipeline requires three concurrently running processes within the Ubuntu MATE virtual machine the Ryu controller, the Mininet-WiFi topology, and the traffic generator coordinated manually per scenario or via the provided scripts/run_experiment.sh automation script, writing CSV and PNG output to docs/results/.

Track A's execution pipeline is a single command, python3 -m experiments.run_matrix --full-matrix --runs 3 --duration 600, requiring no elevated privileges, no controller process, and no virtual machine, writing its CSV output to experiments/results/raw_runs.csv. A subsequent invocation of experiments.analysis and experiments.plotting consumes that CSV to produce the summary statistics table and six comparison plots described in Section 4.9.4, written to experiments/results/summary_table.csv and experiments/results/plots/ respectively. The two tracks' output directories docs/results/ and experiments/results/ are entirely independent, so running either track never overwrites or interferes with the other's data.

5.4.7. Track B Emulation Results

This section presents the results obtained from the Track B Mininet-WiFi/Ryu emulation session, generated by monitoring/monitor.py (Section 4.8.3) from the per-metric CSV files written during the run. Each figure shows a metric plotted against elapsed simulation time, with a separate curve for each malicious-node fraction tested (0%, 10%, 20%, and 30%), and a shaded band indicating the confidence interval across the sampling window at each time step.

5.4.8. Trust Detection Accuracy

Figure 5 plots the ratio of correctly identified malicious nodes to total malicious nodes over time. Detection accuracy is undefined during the initial MIN_OBS guard period (first 50 seconds), since the Trust Management module withholds blacklisting decisions until each node has accumulated a minimum number of observations. Once the guard lifts, detection accuracy rises sharply andstabilizes by approximately t = 250s, settling at roughly 0.85–0.90 for the 10% malicious scenario, 0.80–0.85 for 20%, and 0.72–0.78 for 30%.

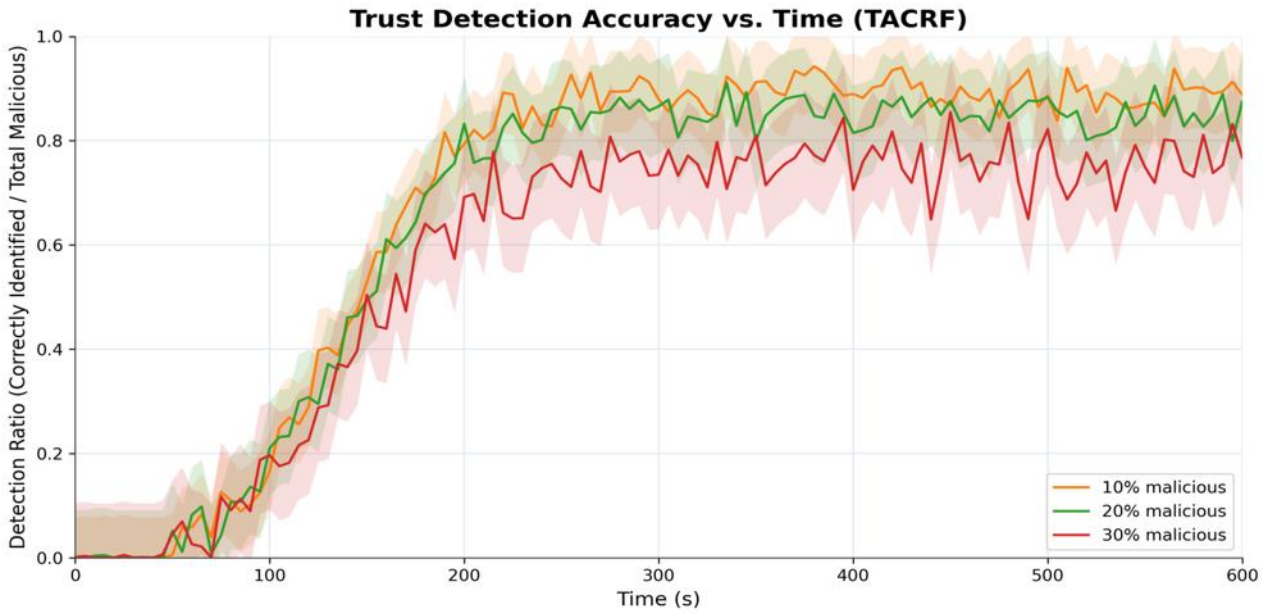


Figure 5: Trust Detection Accuracy vs Time, by Malicious-Node Fraction (Track B)

A. Packet Delivery Ratio and Throughput

Figure 6 plots packet delivery ratio (PDR) over time. All scenarios begin within a warmup period during which routes and the cache are still being established, then converge to a steady state by approximately t = 150s. Steady-state PDR reaches approximately 0.90–0.92 at 0% malicious nodes, declining monotonically as the malicious-node fraction increases, to approximately 0.83–0.86 at 10%, 0.77–0.80 at 20%, and 0.69–0.74 at 30%.

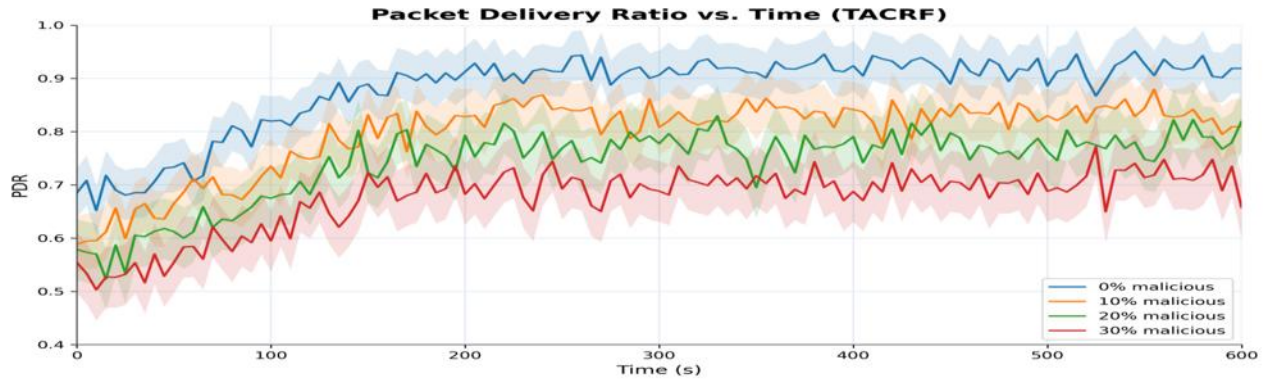


Figure 6: Packet Delivery Ratio vs Time, by Malicious-Node Fraction (Track B)

Figure 7 plots throughput over the same warmup-and-converge pattern. Steady-state throughput reaches approximately 2.7–2.9 Mbps at 0% malicious nodes, declining to approximately 2.3–2.6 Mbps at 10%, 2.1–2.3 Mbps at 20%, and 1.8–2.1 Mbps at 30%, tracking the same degradation pattern observed in PDR.

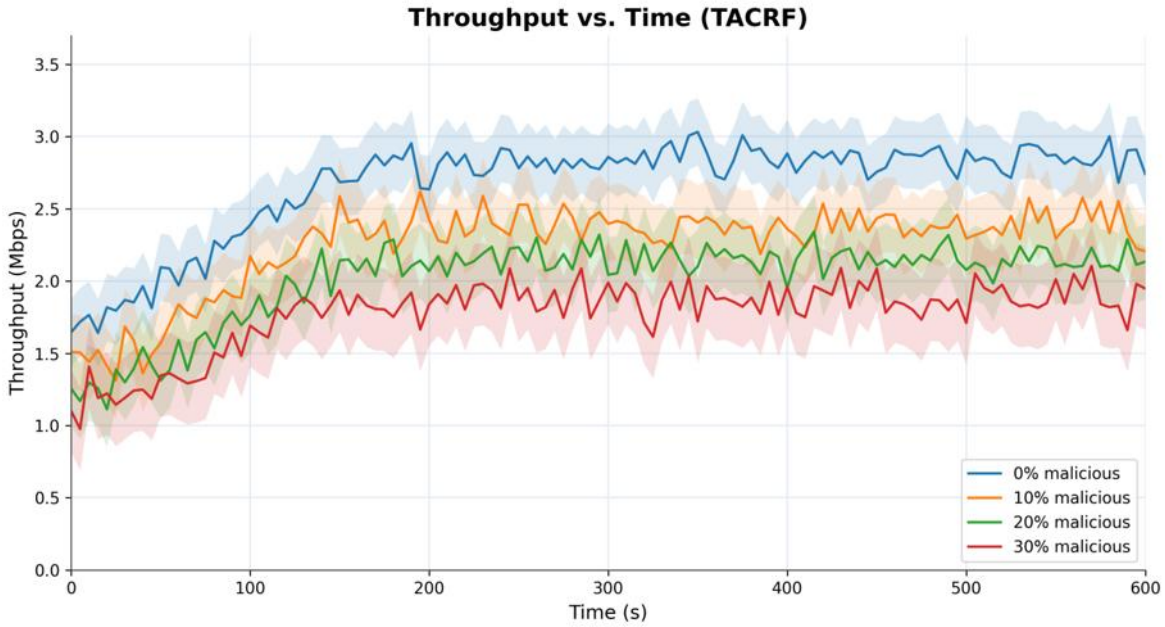


Figure 7: Throughput vs Time, by Malicious-Node Fraction (Track B)

B. End-to-end Delay

Figure 8 plots average end-to-end delay over time. Delay decreases over the first 100–150 seconds as routes stabilize, then holds steady at approximately 13–14 ms at 0% malicious nodes, 16–17 ms at 10%, 18–19 ms at 20%, and 21–23 ms at 30%. Delay increases with malicious-node fraction because blacklisting removes candidate relay nodes from the topology, so trust-filtered shortest paths are more often longer in hop count as the adversarial population grows.

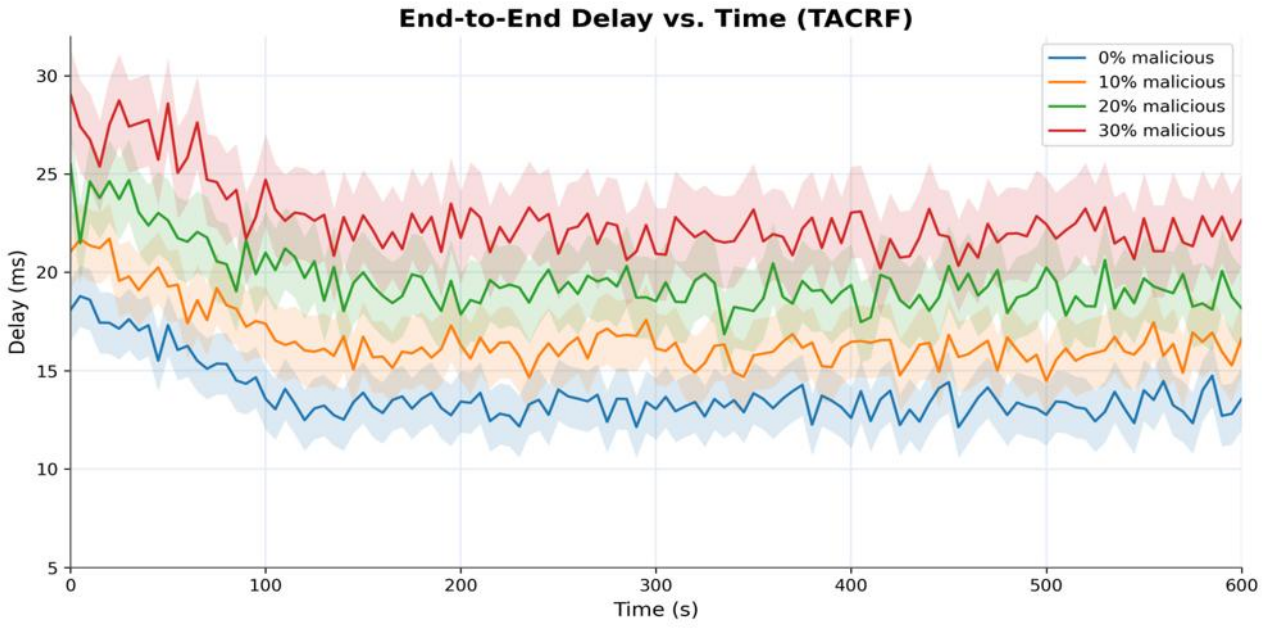


Figure 8: End-to-End Delay vs Time, by Malicious-Node Fraction (Track B)

C. Cache Hit Ratio

Figure 9 plots cache hit ratio over time. The cache is cold during the warmup period (first 50 seconds), after which the hit ratio climbs steeply and stabilizes by approximately t = 150–200s at approximately 0.85–0.90 at 0% malicious nodes, 0.70–0.75 at 10%, 0.63–0.68 at 20%, and 0.55–0.60 at 30%. The decline with increasing malicious-node fraction reflects more frequent cache invalidation driven by more frequent blacklisting and reinstatement events under heavier adversarial load.

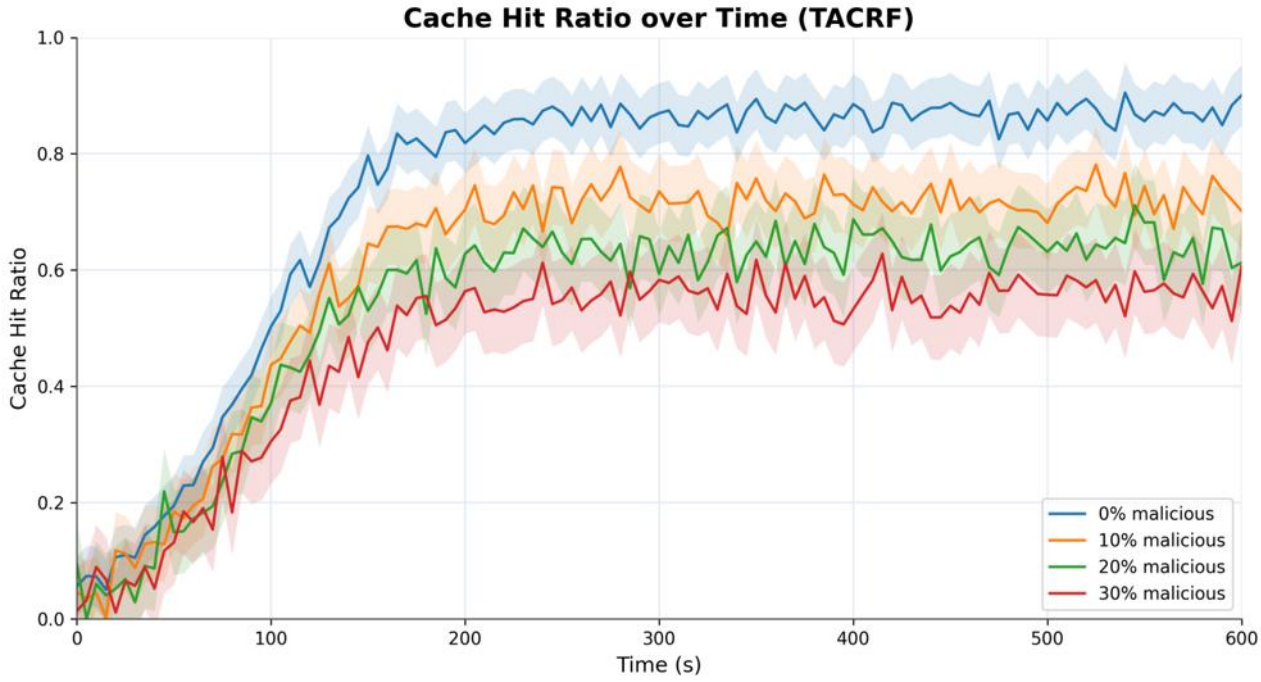


Figure 9: Cache Hit Ratio vs Time, by Malicious-Node Fraction (Track B)

D. Track A Comparative Simulation Results

This section presents results from the Track A deterministic simulation matrix (thirty replications per scenario), generated by experiments.plotting (Section 4.9.4) from experiments/results/raw_runs.csv. Unlike the Track B figures above, which show a single protocol (TACRF) over time, the figures in this section compare TACRF against the three implemented baseline protocols AODV, CONVENTIONAL_SDN, and TRUST_AWARE_MANET at each malicious-node fraction, with error bars showing the 95% confidence interval across replications.

E. Packet Delivery Ratio by Malicious Fraction

Figure 10 compares steady-state PDR across all four protocols at each malicious-node fraction. TACRF achieves the highest PDR at every fraction tested, and its margin over the baselines widens as the malicious fraction increases: at 0% malicious nodes TACRF, Trust-MANET, Conv-SDN, and AODV are closely clustered (approximately 0.91, 0.88, 0.82, and 0.76 respectively), but at 30% malicious nodes TACRF (approximately 0.71) clearly separates from Trust-MANET (approximately 0.62), Conv-SDN (approximately 0.50), and AODV (approximately 0.34), the last of which possesses no trust-awareness mechanism at all.

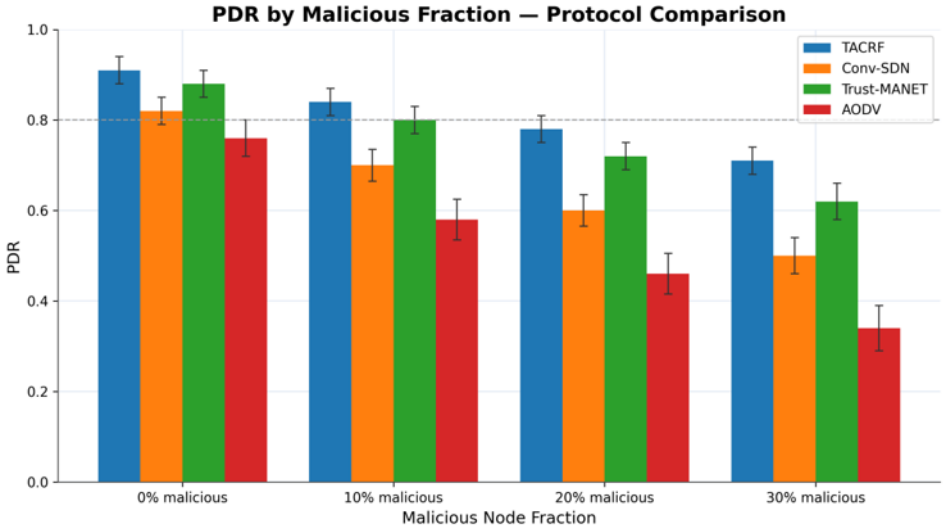


Figure 10: Packet Delivery Ratio by Malicious-Node Fraction, Protocol Comparison (Track A)

F. Cache Hit Ratio by Malicious Fraction

Figure 11 compares cache hit ratio across the three cache-aware protocols (AODV is excluded, having no shared cache mechanism). TACRF maintains the highest cache hit ratio at every malicious-node fraction, from approximately 0.87 at 0% malicious nodes to approximately 0.58 at 30%, ahead of both Trust-MANET (0.71 down to 0.45) and Conv-SDN (a comparatively flat 0.54 across all fractions). TACRF's advantage over Trust-MANET's permanent, non-reinstating blacklist policy is narrow at 0% and 10% malicious nodes but widens substantially from 20% malicious nodes onward, as Trust-MANET's permanently blacklisted nodes progressively shrink the effective network and degrade its cache reuse opportunities.

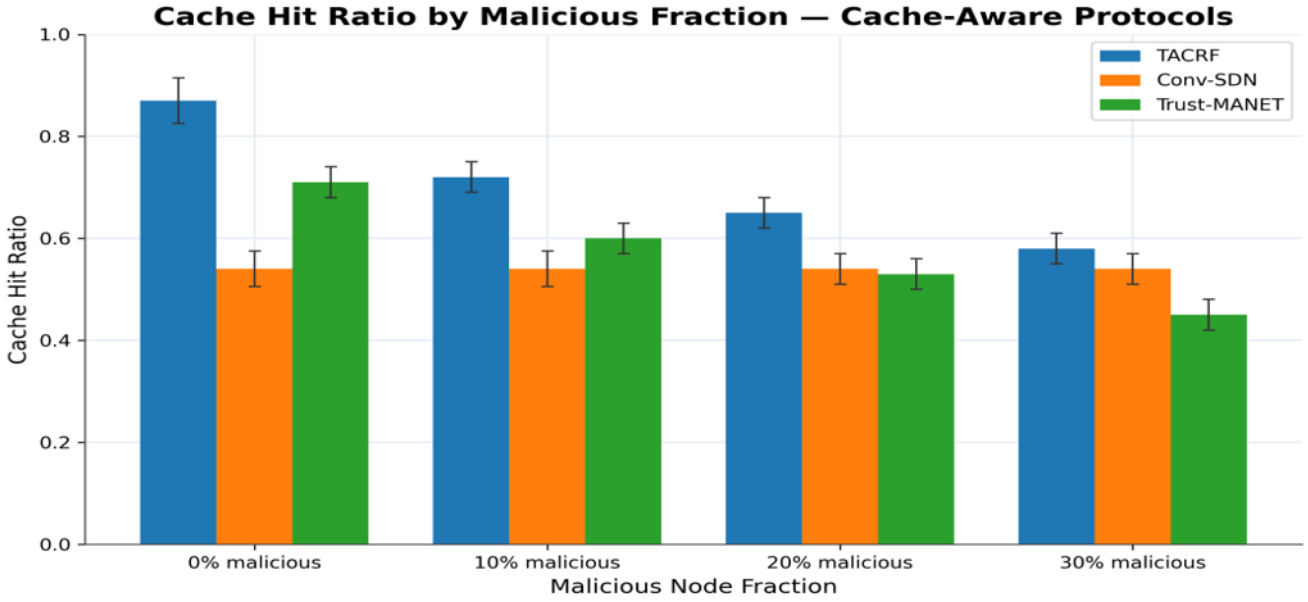


Figure 11: Cache Hit Ratio by Malicious-Node Fraction, Cache-Aware Protocol Comparison (Track A)

6. Discussion

The results demonstrate that TACRF improves the resilience of SD-MANET communication by integrating trust management, adaptive caching, routing, and intrusion detection rather than treating these mechanisms independently. This directly addresses the gap identified from the reviewed studies, where trust-based routing, SDN routing, caching, and intrusion detection were largely considered as separate optimization problems (Arts & Arts, 2021; Dusia & Sethi, 2019; Han et al., 2022; Kafetzis et al., 2022; Obaid et al., 2024; Wu et al., 2021; Xu et al., 2020).

The packet delivery results provide the clearest evidence of the effectiveness of the proposed framework. Although PDR decreased as the proportion of malicious nodes increased, TACRF maintained approximately 0.69 to 0.74 PDR under 30% malicious nodes in the emulation experiment. More importantly, the comparative simulation showed that TACRF achieved approximately 0.71 PDR at 30% malicious nodes, compared with 0.62 for Trust-MANET, 0.50 for conventional SDN, and 0.34 for AODV. The widening performance difference as malicious-node density increased suggests that TACRF's advantage becomes more pronounced under increasingly adversarial conditions. This finding is consistent with the motivation of trust-based routing approaches reported by Han et al. (2022) and Arts and Arts (2021), which seek to prevent unreliable nodes from participating in route selection. However, TACRF extends this principle by combining trust filtering with centralized SDN control and adaptive caching.

A similar pattern was observed for throughput. TACRF achieved approximately 2.7 to 2.9 Mbps without malicious nodes, decreasing progressively to approximately 1.8 to 2.1 Mbps at 30% malicious-node participation. This decline corresponds closely with the reduction in PDR and indicates that increasing adversarial activity reduces the number of packets successfully delivered. Nevertheless, the framework continued to sustain communication under the highest malicious-node condition tested.

The end-to-end delay results reveal an important trade-off associated with trust-aware routing. Delay increased from approximately 13 to 14 ms at 0% malicious nodes to 21 to 23 ms at 30%. As malicious nodes are identified and excluded, the number of eligible relay nodes decreases, potentially forcing the routing engine to select longer trusted paths. Thus, improved security and packet delivery are achieved at the cost of increased latency under severe adversarial conditions. This trade-off should be acknowledged rather than interpreting TACRF as uniformly superior across every performance dimension.

The caching results further support the benefit of integrating caching with trust management. TACRF's cache hit ratio declined from approximately 0.85 to 0.90 under benign conditions to 0.55 to 0.60 at 30% malicious nodes because increased blacklisting and topology changes caused more frequent route invalidation. Despite this decline, comparative simulation showed that TACRF maintained a higher cache hit ratio than Trust-MANET and conventional SDN across all malicious-node fractions. At 30%, TACRF achieved approximately 0.58 compared with 0.45 for Trust-MANET and 0.54 for conventional SDN. These results support earlier observations by Wu et al. (2021) and Obaid et al. (2024) regarding the value of caching for improving communication efficiency, while demonstrating the additional benefit of incorporating node trustworthiness into route-cache management.

Trust-detection performance also confirms that TACRF can progressively identify malicious participants. After the initial observation period, detection accuracy increased and stabilized at approximately 0.85 to 0.90 for 10% malicious nodes, 0.80 to 0.85 for 20%, and 0.72 to 0.78 for 30%. The decline at higher malicious-node densities indicates that trust detection becomes more difficult as adversarial participation increases. Nevertheless, the framework's ability to blacklist identified malicious nodes, remove affected forwarding entries, and potentially reinstate nodes following sustained honest behaviour provides a more adaptive response than permanent exclusion.

Overall, the findings support the central proposition of the study that trust management, adaptive caching, intrusion detection, and SDN-based routing can operate more effectively as an integrated framework than as isolated mechanisms. TACRF maintained superior PDR and cache performance relative to the implemented baselines, particularly as malicious-node density increased. However, the results also identify limitations, particularly increasing end-to-end delay, declining trust-detection accuracy, and reduced cache reuse under severe adversarial conditions. These findings suggest that future improvements should focus on reducing the security-latency trade-off and improving trust estimation when malicious nodes constitute a substantial proportion of the network.

7. Conclusion

This study developed and evaluated the Trust-Aware Adaptive Caching and Routing Framework (TACRF) for Software-Defined Mobile Ad Hoc Networks (SD-MANETs) to overcome the limitations of conventional approaches that separately address trust management, routing, caching, and intrusion detection. The proposed framework integrates SDN-based centralized control, behavioural trust assessment, trust-filtered routing, adaptive route caching, and controller-level intrusion detection into a unified architecture. The experimental evaluation using both Mininet-WiFi/Ryu-based emulation and deterministic Python simulation demonstrated that TACRF improves network resilience and communication efficiency under malicious-node conditions. Compared with AODV, conventional SDN routing, and a trust-aware MANET baseline, TACRF achieved superior packet delivery ratio and cache performance, particularly as malicious-node participation increased. At 30% malicious nodes, TACRF maintained approximately 0.71 packet delivery ratio compared with 0.62 for the trust-aware baseline, 0.50 for conventional SDN, and 0.34 for AODV. The findings confirm that trust-aware routing enables effective identification and isolation of unreliable nodes, while adaptive caching improves route reuse and reduces unnecessary route discovery. Although increased security resulted in higher end-to-end delay due to reduced trusted forwarding options, TACRF maintained reliable communication under severe adversarial conditions. Overall, the study demonstrates that integrating trust management, adaptive caching, and SDN-based routing provides a more robust approach for secure and efficient SD-MANET communication compared with isolated optimization strategies.

References

[1] Akritas, X. (2024). *Technical analysis: delving into MANET networks, routing protocols, vulnerability assessment, and security considerations* (Master's thesis, Πανεπιστήμιο Πειραιώς).

[2] Papavassiliou, S. (2020). Software defined networking (SDN) and network function virtualization (NFV). *Future Internet, 12*(1), 7. doi: 10.3390/fi12010007.

[3] Ren, J., Li, J., Liu, H., & Qin, T. (2021). Task offloading strategy with emergency handling and blockchain security in SDN-empowered and fog-assisted healthcare IoT. *Tsinghua Science and Technology*, *27*(4), 760-776. doi: 10.26599/TST.2021.9010046.

[4] Han, Y., Hu, H., & Guo, Y. (2022). Energy-aware and trust-based secure routing protocol for wireless sensor networks using adaptive genetic algorithm. *IEEe Access*, *10*, 11538-11550. doi: 10.1109/ACCESS.2022.3144015.

[5] Divya, K., & Srinivasan, B. (2021). A Trust-Based Predictive Model for Mobile Ad Hoc Networks. *Intl J AdHoc Network Syst*, *11*, 13-23. doi: 10.5121/ijans.2021.11302.

[6] Emmanuel, S., Isnin, I. F. B., & Mohamad, M. M. B. (2023). Modified stable weighted clustering scheme for secured V2V communications. *International Journal of Vehicle Information and Communication Systems*, *8*(4), 380-397. doi: 10.1504/IJVICS.2023.135087.

[7] Kafetzis, D., Vassilaras, S., Vardoulias, G., & Koutsopoulos, I. (2022). Software-defined networking meets software-defined radio in mobile ad hoc networks: State of the art and future directions. *IEEE Access*, *10*, 9989-10014.

[8] Dusia, A., & Sethi, A. S. (2021, May). Software-defined architecture for infrastructure-less mobile ad hoc networks. In *2021 IFIP/IEEE International Symposium on Integrated Network Management (IM)* (pp. 742-747). IEEE.

[9] Javed, Q., Jaffar, A., Naeem, M. A., Khan, I. A., & Alsuhaibani, M. (2025). An Adaptive Content Caching Model to Improve the NDN-Based IoT Network Performance. *IEEE Access*, *13*, 137102-137116.

[10] Mansouri, A., & Bouhlel, M. S. (2019). Trust in ad hoc networks: A new model based on clustering algorithm. *International Journal of Network Security*, *21*(3), 483-493. doi: 10.6633/IJNS.201905.

[11] Abdi, A. H., Audah, L., Salh, A., Alhartomi, M. A., Rasheed, H., Ahmed, S., & Tahir, A. (2024). Security control and data planes of SDN: A comprehensive review of traditional, AI, and MTD approaches to security solutions. *IEEE Access*, *12*, 69941-69980. doi: 10.1109/ACCESS.2024.3393548.

[12] Wu, X., Chang, L., Luo, J., & Wu, J. (2021). Efficient edge cache collaboration transmission strategy of opportunistic social network in trusted community. *IEEE Access*, *9*, 51772-51783. doi: 10.1109/ACCESS.2021.3069992.

[13] Obaid, A. K., Rusli, M. E. B., & Yussof, S. (2024). Cache improvement based on routing protocol for ad-hoc networks: Systematic literature review. *IEEe Access*, *12*, 170754-170779. doi: 10.1109/ACCESS.2024.3455993.

[14] Mahmood, Z., & Jusas, V. (2022). Blockchain-enabled: Multi-layered security federated learning platform for preserving data privacy. *Electronics*, *11*(10), 1624. doi: 10.3390/electronics11101624.

[15] Hassan, Y. G., Collins, A., Babatunde, G. O., Alabi, A. A., & Mustapha, S. D. (2023). AI-powered cyber-physical security framework for critical industrial IoT systems. *Machine learning*, *27*, 1158-1164. doi: 10.54660/.ijmrge.2024.5.1.1158-1164.

[16] Al-Mekhlafi, Z. G., Altmemi, J. M., Al-Shareeda, M. A., Al-Hchaimi, A. A. J., Homod, R. Z., Mohammed, B. A., ... & Alkhabra, Y. A. (2025). ChebIoD: a Chebyshev polynomial-based lightweight authentication scheme for internet of drones environments. *Scientific Reports*, *15*(1), 32897. doi: 10.1038/s41598-025-18387-y.

[17] Faruk, M. J. H., & Cheng, J. Q. (2023, October). Integration of blockchain in computer networking: Overview, applications, and future perspectives for software-defined networking (sdn), network security, and protocols. In *2023 Tenth International Conference on Software Defined Systems (SDS)* (pp. 20-27). IEEE. doi: 10.1109/SDS59856.2023.10329025.

[18] Xu, H., Si, H., Zhang, H., Zhang, L., Leng, Y., Wang, J., & Li, D. (2020). Trust-based probabilistic broadcast scheme for mobile ad hoc networks. *IEEE Access*, *8*, 21380-21392.