computers

MDPI

Article

R-Snort: A Performance-Optimized Multi-Agent NIDS Architecture for SOHO and Edge-of-Things Networks Using Snort 3 on Raspberry Pi 5

Julio Gómez López [1,*], Deian Orlando Petrovics Tabacu [1], Nicolás Padilla Soriano [1] and Alfredo Alcayde García [2]

[1] Department of Computer Science, University of Almería, Ctra Sacramento, s/n, 04120 Almería, Spain; dp189@inlumine.ual.es (D.O.P.T.); npadilla@ual.es (N.P.S.)
[2] Department of Engineering, University of Almería, Ctra Sacramento, s/n, 04120 Almería, Spain; aalcayde@ual.es
* Correspondence: jgomez@ual.es

Abstract

Network Intrusion Detection Systems (NIDSs) are critical to ensuring the resilience of modern digital infrastructures. Although traditionally deployed in large-scale corporate environments, the expanding threat landscape requires the integration of robust security measures into Small Office/Home Office (SOHO) and Edge-of-Things (EoT) networks. However, these environments often face significant constraints in terms of specialized hardware and technical expertise. This article presents R-Snort, an open-source NIDS based on Snort 3, optimized for low-cost Raspberry Pi 5 hardware. Its multi-agent architecture enables distributed deployment with centralized traffic analysis and cross-agent attack correlation, while an intuitive web interface simplifies alert visualization and system management for non-expert administrators. Its main contributions are: (1) a performance-optimized NIDS agent achieving 1 Gbps throughput; (2) a distributed multi-agent architecture enabling centralized event correlation and detection of multi-vector attacks; and (3) an IaC-based automated deployment framework with an intuitive web interface, democratizing professional-grade security for SOHO and EoT environments.

Keywords: Snort; network intrusion detection systems; multi-agent systems; raspberry Pi; edge computing security; SOHO network security; edge-of-things (EoT) environment

1. Introduction

Network Intrusion Detection Systems (NIDSs) are a fundamental cybersecurity tool that analyzes all network traffic in real time to detect and analyze cyberattacks [1]. Although their use has traditionally focused on large-scale corporate infrastructures, SOHO networks are a growing target for increasingly sophisticated cyberattacks, driven by the expansion of remote work and the reliance on cloud services [2]. This reality underscores the need to democratize IDS/IPS solutions for environments with limited resources.

In this context, Snort [3] is one of the most prevalent and versatile open-source NIDSs, widely used for its ability to perform real-time traffic analysis and attack detection [4]. Although its use has historically been associated with high-performance servers, the evolution of technology allows its adaptation to lower-cost hardware, with the aim of extending the use of NIDSs.

Check for updates

Academic Editor: Paolo Bellavista

Received: 1 March 2026
Revised: 19 April 2026
Accepted: 20 April 2026
Published: 24 April 2026

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

This article presents R-Snort, a comprehensive open-source NIDS based on Snort 3, which has been optimized to run on low-cost hardware like the Raspberry Pi 5. R-Snort includes an intuitive web interface and a multi-node architecture, which enables the distributed deployment of agents to centralize event correlation and traffic analysis. This low-cost approach and multi-node architecture facilitate deployment and scalability and make it possible to democratize cybersecurity in resource-constrained environments. The main contributions of this research are summarized as follows:

- **Performance Optimization:** A multi-agent NIDS designed specifically for Raspberry Pi 5, achieving enterprise-level performance on affordable hardware. Its use is essential for improving the security of SOHO and Edge-of-Things networks.
- **Usability-Security Convergence:** The development of an intuitive management web and real-time dashboards, specifically designed to bridge the operational gap for non-expert administrators.
- **Automated Deployment:** R-Snort deploys in minutes using automated scripts, requiring no specialized Linux or networking knowledge. This makes enterprise-grade NIDS accessible to small business administrators.
- **Distributed Visibility:** Multiple R-Snort agents can be deployed across different network segments, all reporting to a single dashboard. This allows detection of coordinated attacks that a single-point sensor would miss.

2. Related Work

The deployment of Network Intrusion Detection Systems (NIDS) has become a standard security measure in corporate environments. However, most of the current literature focuses on high-speed networks with high-resource infrastructures [5,6], there is a critical need to strengthen security within Small Office/Home Office (SOHO) and Edge-of-Things (EoT) environments. These environments are increasingly targeted by sophisticated cyberattacks but often lack the specialized personnel or budget for enterprise-grade solutions. This technological gap underscores the urgent need for IDS/IPS architectures that are computationally efficient and accessible to resource-constrained hardware.

2.1. NIDS on Low-Cost and Embedded Platforms

Recent academic research has turned toward the analysis of NIDSs on embedded and low-power platforms, moving beyond traditional high-performance server architectures. Snort has established itself as a versatile open-source benchmark capable of real-time traffic analysis through signature-based and anomaly-based detection [7]. While Snort remains a primary focus, studies have increasingly explored alternative engines such as Suricata and Zeek to evaluate their effectiveness and strategic placement in limited hardware environments. Specifically, in [8] analyzes between Snort 3 and Suricata have highlighted advances in modularity and scalability—features vital for accessible hardware. Previous implementations on platforms such as the Raspberry Pi 4 [9] demonstrated the feasibility of using such hardware to identify performance bottlenecks and evaluate the efficacy of NIDS under real-world conditions.

The Raspberry Pi is by far the most widely used platform in academic Snort research, with a substantial body of published work [9–14] providing a direct performance baseline. This makes the Raspberry Pi 5 the natural hardware choice for a contribution seeking comparability with the existing literature. Furthermore, at approximately $150 USD for the 8 GB variant, the RPi5 offers a competitive price point compared to alternative edge computing platforms with similar or greater capabilities.

2.2. Multi-Agent Systems (MAS) and Distributed Detection

A fundamental advantage of Multi-Agent Systems (MAS) is their ability to manage and correlate heterogeneous alerts from distributed sensors [15]. Event correlation is critical; complex attacks often manifest as a sequence of seemingly unrelated events across different network segments. Although a single agent may only detect a fragment of an intrusion, a MAS can reconstruct the full attack scenario through autonomous cooperation and information exchange. This proactive nature allows for more adaptable detection compared to traditional monolithic IDS, which often struggle with scalability [16]. Furthermore, the application of real-time MAS has proven crucial in mitigating processing overheads and improving response latency against emerging threats [17,18]. Beyond detection, intelligent agents are increasingly explored for automated intrusion response (IPS) [19]. By allowing agents to execute corrective actions, the "window of vulnerability" between detection and mitigation is significantly reduced, a critical factor for protecting real-time systems [20]. However, existing Snort-based MAS often focus on combining Host-based IDS (HIDS) agents and frequently lack robust communication protocols to monitor multiple network segments simultaneously [21].

Beyond detection, operator-oriented modeling approaches for distributed dynamic systems [22] present a theoretically grounded perspective that could inspire future enhancements to the analytical capabilities of MAS-IDS event correlation engines.

2.3. Usability Barriers and Alert Interpretation

Despite the technical robustness of engines like Snort, their management and alert interpretation present significant barriers for non-expert users. While traditional interfaces like BASE or ACID provided early solutions, they are now largely deprecated. Contemporary alternatives, such as the ELK stack (Elasticsearch, Logstash, Kibana) [23], offer powerful visualization capabilities but demand high resource overhead and specialized knowledge, making them impractical for SOHO administrators. Consequently, research in this field must prioritize not only technical throughput, but also usability and "plug-and-play" accessibility.

2.4. Advancements and Distinctions of R-Snort

To clarify the scientific contribution and position R-Snort within the existing landscape, Table 1 compares it with the four most closely related prior works across five dimensions: hardware target, IDS engine, agent type, demonstrated throughput, and automated deployment support.

Table 1. Comparison of R-Snort with closely related prior works.

Work	Hardware	IDS Engine	Agent Type	Auto Deploy
Varma et al. [9]	RPi 4B	Snort 2/3	Single-node NIDS	No
Al-Yaseen et al. [17]	Servers	SVM/ELM MAS	Multi-agent (HIDS)	No
Younes & Jemili [16]	Cloud/Spark	Custom MAS	Big-Data oriented	No
Davies et al. [21]	Servers + SIEM	Snort 3 multi-node	Collaborative NIDS	Partial
R-Snort (this work)	RPi 5 (8 GB)	Snort 3 (opt.)	Multi-agent NIDS	Yes (full IaC)

Varma et al. [9] is the closest prior work in terms of platform. However, their implementation uses a single RPi4 node, explicitly documents CPU saturation and RAM bottlenecks under high-traffic conditions, and provides no automated deployment. R-Snort advances this work by migrating to the RPi5 (2–3× the CPU performance of the RPi4's Cortex-A72), introducing a multi-agent architecture, and providing full IaC deployment.

Al-Yaseen et al. [17] propose a real-time MAS using SVM and ELM classifiers evaluated on the KDD'99 benchmark dataset. Their system targets general-purpose servers, is not designed for resource-constrained hardware, and reports no real network throughput.

Younes and Jemili [16] propose a MAS-based IDS on Apache Spark for Big Data environments. It is fundamentally incompatible with the SOHO/EoT context, requiring a Spark cluster and not deployable on embedded hardware.

Davies et al. [21] present the most architecturally similar work: a collaborative IDS using multiple Snort nodes with centralised correlation. However, their architecture targets enterprise environments, requires a commercial LogScale SIEM for visualisation, and has no automated deployment pipeline for non-expert users.

In summary, R-Snort is the only system in this comparison that simultaneously (1) uses Snort 3 on low-cost embedded hardware for SOHO/EoT, (2) implements a true multi-agent NIDS across network segments, (3) demonstrates 1 Gbps throughput with zero packet loss, and (4) provides fully automated deployment requiring no specialised expertise.

3. R-Snort

3.1. Architecture and Implementation

The architecture of R-Snort is modular, distributed, and scalable, designed to adapt to SOHO networks. The system consists of two main elements: the Snort agents—deployed on embedded systems like the Raspberry Pi—and a central module that facilitates comprehensive coordination, management, and monitoring.

Each Snort agent operates as an autonomous sensor, based on a customized version of Snort 3 to optimize performance on the limited hardware of embedded systems like the Raspberry Pi. These agents use a custom Snort installation compiled from its source code, incorporating various libraries for proper functioning, such as libdnet, libdaq, LuaJIT, Flex, Bison, and PCRE2 (see Figure 1). This allows for efficient and stable system execution on the Raspberry Pi platform.

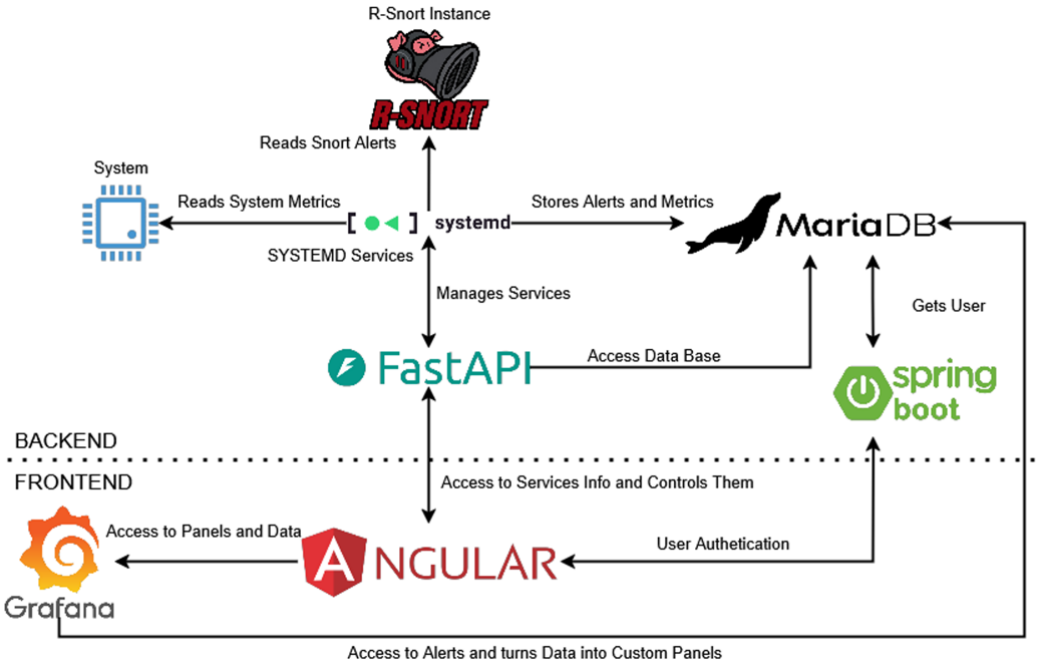


Figure 1. Central Module Architecture Development.

Threat detection is based on a dual approach: official community rules and custom rules specifically developed to identify sensitive data (e.g., credentials or bank cards), thereby compensating for the absence of the old "Sensitive Data" preprocessor, which has

been discontinued in Snort 3. Additionally, the system uses various Snort preprocessors, such as HTTP Inspect, SSL Inspector, Stream IP, Stream TCP, and Reputation, which provide an exhaustive inspection of traffic to detect evasion techniques and encrypted threats [3]. Furthermore, it used the ClamAV antivirus module which provides an additional layer of security against malware through signature-based analysis.

The centralization of traffic and attack correlation is achieved by consolidating data from multiple agents in the central module. For this purpose, the information of each agent is stored in real-time in a MariaDB database using auxiliary services: *ingest_service.py* (for alerts) and *metrics_timer.py* (for system metrics such as CPU, RAM, and temperature).

The central module, built on a Spring Boot (Java 17) backend and an Angular 19 frontend, acts as the central management and visualization point for the network. Its backend exposes a secure REST API (protected with JWT) that allows the web interface to remotely manage agents, query alerts, manipulate rules, and control internal services. The frontend offers a modular interface with advanced functionalities such as remote rule management, alert downloading, and integrated dashboard visualization via Grafana.

Communication between agents and the central module is carried out exclusively through a REST API, implemented with FastAPI. This design, which uses HTTP/HTTPS protocols and JSON data format, not only optimizes network resource consumption and ensures data security but also guaranties that each agent can operate independently in case of a temporary disconnection from the central module. The REST API exposes various endpoints for information exchange, such as */alerts*, */status*, */rules*, and */download-alerts*.

The Snort 3 compilation was performed with optimization flags (*-O2*) and hardware-specific tuning for the ARM Cortex-A76 architecture. Key configuration parameters include: *max_pkt_len=1518*, *thread_count=4* (one per CPU core), and *stream5_global.track_tcp=yes*.

3.2. Deployment

The design of R-Snort focuses on ease of deployment, offering a plug-and-play experience ideal for environments with minimal specialized personnel. This approach, inspired by Infrastructure as Code (IaC), fully automates the system's installation and configuration, ensuring an efficient and secure implementation even for users with limited system administration knowledge.

The R-Snort architecture is structured into three independent components that simplify the process:

- **rsnort-installer**: Completely installs Snort 3, including its dependencies, preprocessor configuration, and the activation of advanced detection rules.
- **snort-agent**: Installs a monitoring agent, integrating essential services such as MariaDB, Grafana, metrics collection, log rotation, and a REST API for system interaction.
- **rsnort-central-module**: Installs the administration interface, which includes the Spring Boot backend and the Angular frontend.

The entire process is automated using self-running installers, designed to execute without user intervention (except for basic network configuration and administrator credentials), which facilitates the installation. Additionally, the project includes technical documentation on GitHub https://github.com/jugomezual/R-Snort (accessed on 19 April 2026), accessible from the official repository [24], that guides the user step-by-step through the deployment and use of the system.

3.3. Administration Frontend

The web interface of the central R-Snort module was developed using Angular 19 as the main framework, using its capabilities to build a modular, reactive, and intuitive application. The interface is divided into specific modules that allow the visualization of the system's

status and general dashboard (see Figure 2), alert management, rule administration, and agent administration, among other modules.



Figure 2. Main Dashboard.

To improve and provide real-time analysis, personalized Grafana dashboards have been integrated directly into the interface. For this task, the "Snort IDS/IPS Dashboard" [25] was taken as inspiration. These dashboards allow the administrator to get a quick and clear view of the current status of the system, key performance metrics, and patterns in the detected alerts.

In the Status tab, system metrics such as CPU, disk and temperature are displayed, along with the current Snort status, the last alert received, and options to restart the Snort, storage, and metrics services (see Figure 3).



Figure 3. Snort Status.

The visualization of alerts on the frontend is dynamic and interactive, showing alerts by different categories like severity, alert type, and attack origin, among others (see Figure 4).

The system allows for the administration of all active rules and the creation of new rules. Additionally, it enables the export of alerts in CSV format, just as it allows the download of compressed files with archived alerts to facilitate later analysis and forensic tasks.

In the case of using the multi-agent system, the top menu allows you to see the different agents (initially only the central module appears), and new agents can be added at any time. Once added, the agent appears in the dropdown menu, and the dashboard automatically updates with its metrics and alerts. In addition, active agents are shown in the Status view, and it is possible to manage services and visualize the last detected alert (see Figure 5).

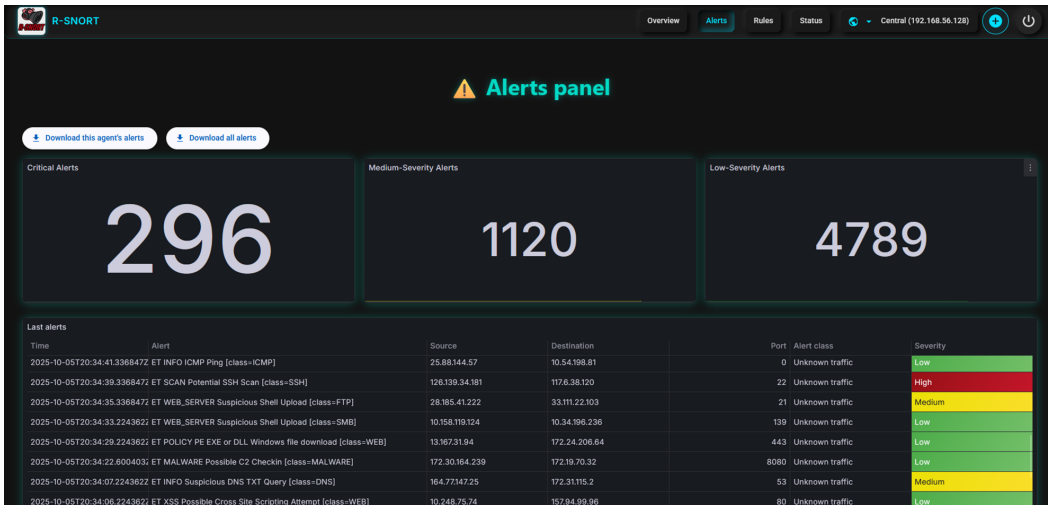


Figure 4. Alerts Panel.



Figure 5. Agent Status.

3.4. Security Considerations

The central node and secondary nodes are protected by their network architecture: the interface that listens to network traffic operates in monitor (promiscuous) mode, through which no system is directly accessible from the monitored network. System administration and inter-agent communication are handled exclusively through a separate internal network interface on a dedicated intranet, isolated from external access. This network separation provides a fundamental layer of protection by design, ensuring that the NIDS nodes expose no attack surface to the traffic they are monitoring.

Agent-to-central communication over the administration intranet is further secured using JWT-authenticated REST API calls over HTTPS, ensuring data integrity and confidentiality in transit. Individual agents are designed to operate fully autonomously in the event of a temporary or permanent disconnection from the central module, buffering alerts locally to prevent data loss and resuming synchronisation once connectivity is restored.

Known limitations of the current architecture include: the central correlation node constitutes a logical single point of failure for cross-agent event correlation (mitigated by the autonomous operation of individual agents); the current implementation does not enforce mutual TLS for agent authentication; and the signature-based detection engine remains susceptible to known evasion techniques targeting Snort rule sets. A formal adversarial security analysis of the full architecture is identified as a critical direction for future work.

4. Evaluation

To validate the effectiveness of R-Snort, a case study was built to replicate a small business network environment. The system was put into operation on a local network that uses a manageable switch to implement Port Mirroring functionality and send all network traffic to the Raspberry Pi 5. This approach ensures that R-Snort receives and exhaustively analyzes all network traffic (see Figure 6).

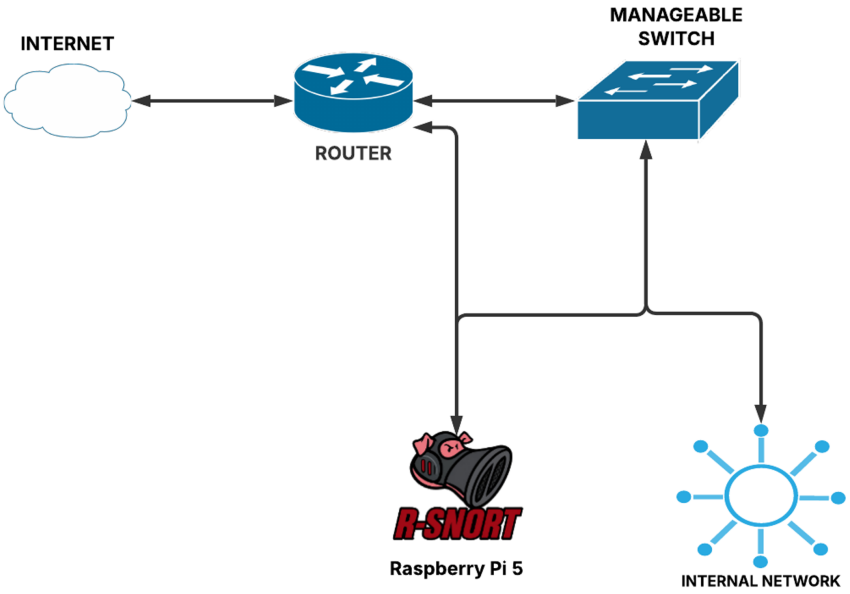


Figure 6. Work Environment.

R-Snort requires a minimum of 4 GB RAM (8 GB recommended), an ARMv8 or x86_64 processor, Raspberry Pi OS Bookworm or Debian 12, and a network interface supporting promiscuous mode.

To make the test network, the following devices were used:

- A router connected to the switch, which provides Internet access.
- A manageable switch (**Tenda TEG205E**, Tenda Technology Co., Ltd., Shenzhen, China) with the ability to replicate traffic from one port to another.
- Specifically, three Windows 11 workstations and two Ubuntu 22.04 laptops connected to the switch, simulating typical office traffic.
- A **Raspberry Pi 5** (Raspberry Pi Ltd., Cambridge, UK) running R-Snort, with the following key specifications: CPU: ARM Cortex-A76 (4 cores at 2.4 GHz); RAM: 8 GB LPDDR4X; Storage: A 32 GB microSD card.

In this way, the manageable switch sends all network traffic generated by the company to the Raspberry Pi so that Snort can analyze the traffic in search of any type of attack.

4.1. Test 1: Performance Evaluation

To validate the efficiency of R-Snort, performance tests were conducted to evaluate its ability to operate stably on a Raspberry Pi under high workloads. The primary objective was to determine the system's behavior under both normal conditions and a stress test with 1 Gbps of traffic (the maximum allowed in a typically SOHO network) without compromising system stability and efficiency. The 1 Gbps stress traffic was generated using `tcpreplay` replaying a captured PCAP file at the maximum supported line rate, and independently verified using `iperf3` between two hosts on the test network. For normal conditions, the test period consisted of: 0–60 s idle, 60–600 s normal office traffic (web browsing, email, file transfers), 600–660 s idle. For the stress test, traffic was injected at 1 Gbps for 300 s.

The results obtained are as follows.

- **CPU Usage Analysis**: Under normal conditions, CPU usage is minimal. The application time (cpu.user) remains between 0.25% and 0.75%, while the kernel-level time (cpu.sys) is negligible, leaving cpu.idle above 98%. However, during the stress test, CPU usage increases significantly: cpu.user reaches 20–25% and cpu.sys 55–60%, resulting in a total active CPU consumption of approximately 80%, with cpu.idle dropping to 10–15%. Despite this high demand, the system demonstrates a great capacity to process the stress test load without saturation or thermal throttling, which validates the effectiveness of R-Snort (see Figure 7). Figure 7 shows normal conditions on the left panel and the stress test on the right panel, enabling direct comparison. Red denotes user-space CPU usage (cpu.user), reflecting R-Snort's packet inspection workload; blue denotes kernel-level CPU usage (cpu.sys); yellow denotes idle CPU (cpu.idle); the x-axis shows wall-clock time. The dashed vertical line separates normal operating conditions (left) from the 1 Gbps stress test (right).

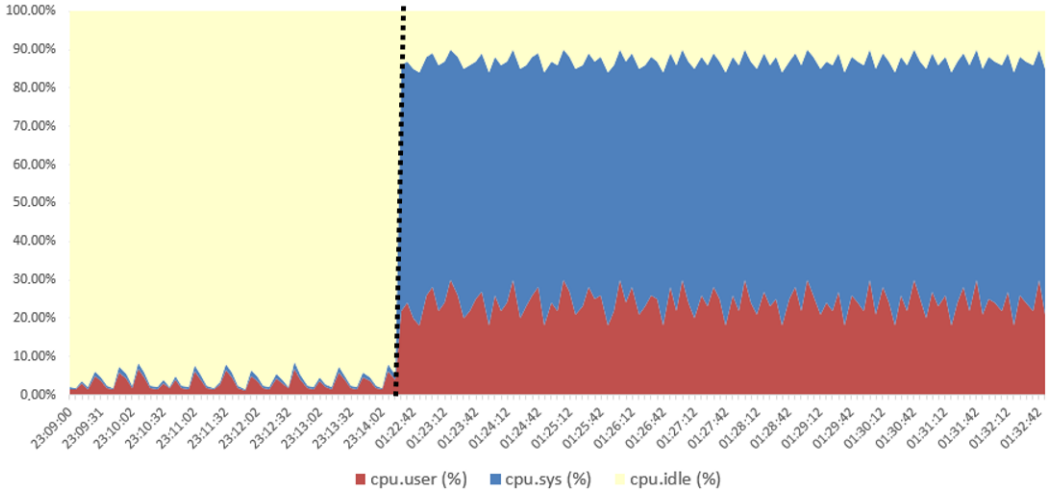


Figure 7. CPU Usage under normal conditions ((**left**) panel) and stress test ((**right**) panel).

- **RAM Memory Analysis**: R-Snort's direct consumption is 2.18 GB of RAM, increasing to 2.8 GB under the stress test. This shows efficient resource management for handling high traffic (see Figure 8). Since RAM usage remains stable under normal conditions, Figure 8 focuses on the stress test phase, where peak consumption is observed. The red line denotes RAM actively used by the system (mem.util.used), which rises gradually from 2.6 GB to approximately 2.8 GB over the course of the test; the dark blue line denotes free RAM (mem.util.free), which decreases correspondingly from approximately 4.6 GB to 4.4 GB; the yellow-green line denotes cached memory (mem.util.cached), which remains flat at approximately 0.3 GB throughout, confirming the absence of cache pressure or memory leaks under sustained load.
- **Disk I/O Analysis**: R-Snort, once the system is started, only performs write operations to log alerts. Since disk activity is negligible under normal conditions, Figure 9 focuses on the stress test phase, where logging bursts are most pronounced. The blue line (r-snort:disk.dev.write_bytes[mmcblk0]) denotes disk write throughput; peaks reflect alert logging bursts triggered by detected events, reaching up to 350 KB/s with approximately 1–2 s duration, after which throughput returns to baseline, confirming that disk I/O does not accumulate or saturate under sustained high traffic. The red line (r-snort:disk.dev.read_bytes[mmcblk0]) remains near zero throughout the entire test, confirming that R-Snort operates as a write-only process once initialised.

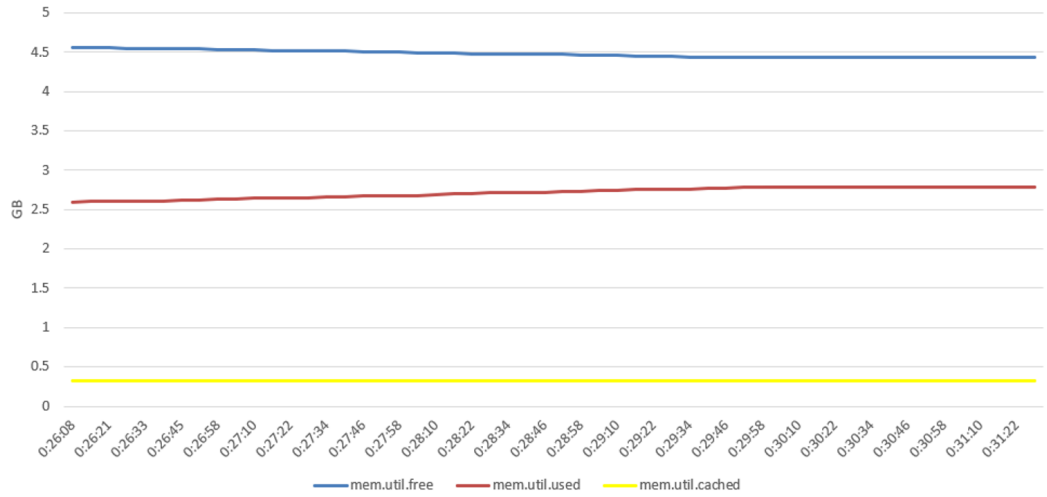


Figure 8. Memory usage under stress test.

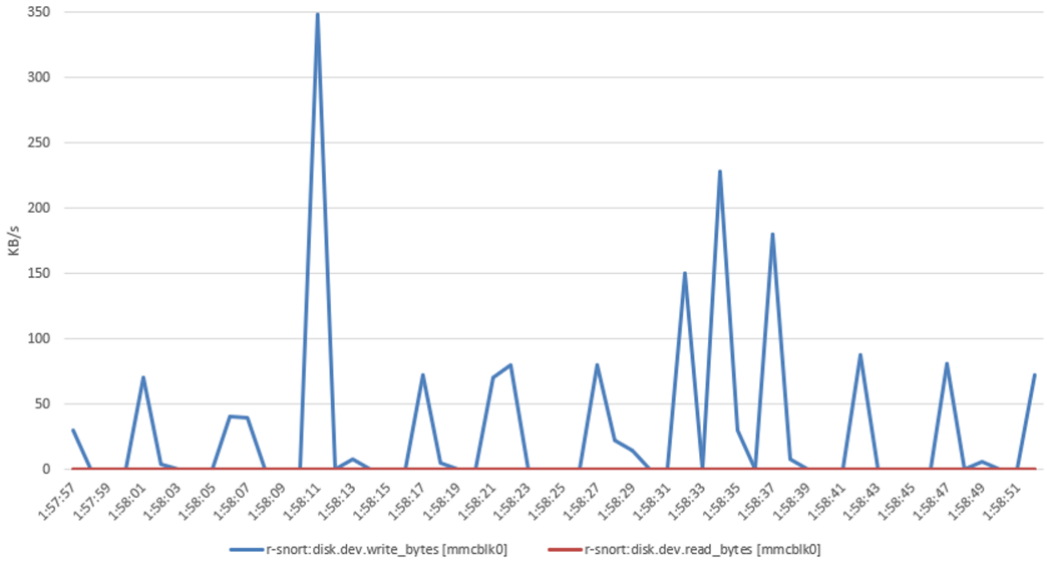


Figure 9. Disk Usage (Snort–Stress Test).

- **Packet Loss Rate**: In both scenarios, R-Snort processed all network traffic with zero dropped packets. This was verified via Snort's built-in performance statistics, which confirmed that 0 packets were dropped out of the total received during the stress tests.

During these tests, the device's temperature remained consistently within the standard operating parameters of the Raspberry Pi (typically below 70 °C). This thermal stability is crucial; it ensures that the system does not trigger thermal throttling, a mechanism where the CPU reduces its clock speed to prevent overheating.

4.2. Test 2: Multi-Agent System Test

The multi-agent architecture of R-Snort was designed to provide network visibility that is not possible to obtain from a single sensor. To validate its operation, a test environment was created with three agents: the Central agent (192.168.1.208), agent AG1 (192.168.1.205), and agent AG2 (192.168.1.207). The analysis focused on the system's ability to detect and correlate events in a coordinated manner, using two key tests:

1. **Multi-Agent Event Correlation**: The first test verified the ability of each agent to analyze the traffic of its network segment and the aptitude of the central module to consolidate alerts. A low-intensity TCP port scan was executed from a test host

directed at the three agents, using common management ports (161, 162 and 705). The scans were executed sequentially (not simultaneously), with a 30-s interval between each scan. As a result, the central module detected and correlated the alerts generated by all sensors, confirming coordinated visibility throughout the network.

2. **Distributed Traffic Analysis**: The next test aimed to confirm that the agents simultaneously detect bursts of SSDP/UPnP discovery traffic directed at the multicast address `239.255.255.250:1900` (see Figure 10). SSDP/UPnP traffic was generated simultaneously from all five client devices over a 60-min period. All corresponding alerts were filtered to measure the number of alerts, the number of agents who reported them, and the number of different origins per minute. The analysis revealed periodic bursts with high-density peaks and concurrent sensor activity. As shown in Figure 10, only agents AG2 (orange, marked with ×) and central (green, marked with •) reported alerts for this traffic pattern, as agent AG1 was deployed on a network segment that did not receive SSDP/UPnP multicast traffic during this test; this is consistent with the expected behaviour of a distributed architecture, where each agent only reports traffic visible from its own network segment. At various times, multiple origins were observed pointing to the same destination, which evidences the system's ability to detect distributed and multi-source attack patterns.

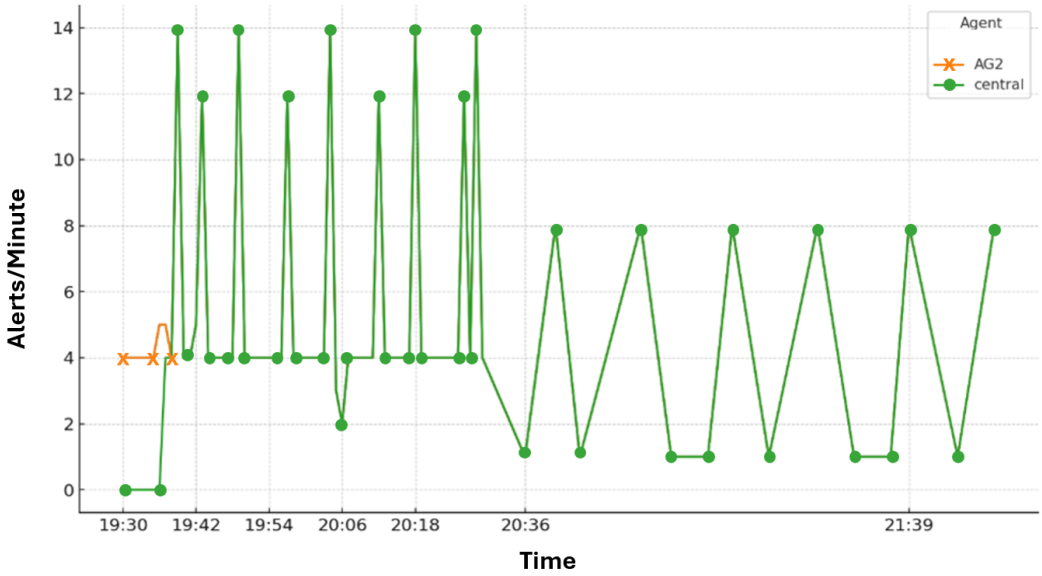


Figure 10. Alerts from the Multi-Agent System—Distributed Traffic Analysis (UPnP).

4.3. Contextual Baseline: Comparison with Prior RPi-Based NIDS

A direct experimental replication on a Raspberry Pi 4 under identical conditions was not performed in the scope of this work. Instead, this section establishes a contextual baseline by comparing R-Snort's documented results against the performance figures reported in the two most relevant prior works that tested Snort on Raspberry Pi hardware under real traffic.

Sforzin et al. [26] (RPiDS) reported that the CPU begins discarding packets at data rates as low as 10 Mbit/s when using short packets on the Raspberry Pi 2. Even with long packets, drops appeared at approximately 70 Mbit/s, and the authors explicitly concluded that a 100 Mbit/s Ethernet interface was inadequate for their platform.

Varma et al. [9] worked with the more capable Raspberry Pi 4B but identified CPU saturation and RAM bottlenecks under high-traffic conditions with Snort, without achieving zero-packet-loss at Gigabit speeds.

In contrast, R-Snort on the Raspberry Pi 5 achieved 0% packet loss at the full 1 Gbps SOHO line rate. Two architectural advances explain this improvement. First, the Cortex-A76 (16 nm, 2.4 GHz) delivers 2–3× the CPU performance of the RPi4's Cortex-A72 (28 nm, 1.5 GHz), with approximately 35% higher IPC [27]. Second, the RPi5 introduces the RP1 southbridge [27], a dedicated I/O controller (PCIe 2.0 ×4) that offloads Ethernet interrupt handling from the main SoC—freeing the CPU for Snort's pattern matching. Since Snort 3 is CPU-bound (rule checking ≈75% of IDS processing time [6]), this I/O offloading directly enables sustained 1 Gbps zero-packet-loss operation. Table 2 summarises the available data points from each source.

Table 2. Contextual performance baseline: R-Snort vs. prior Raspberry Pi + Snort deployments from the literature. NR = not reported by the original authors.

Source	Hardware	Max Throughput (0% Loss)	CPU Under Stress	Packet Loss Onset
Sforzin et al. [26]	RPi 2B	∼70 Mbit/s (long pkts)	100% at 10 Mbit/s (short pkts)	From 10 Mbit/s (short); ∼70 Mbit/s (long)
Varma et al. [9]	RPi 4B (4 GB)	NR—bottlenecks documented	Saturation at high load	Documented; exact rate NR
R-Snort (this work)	RPi 5 (8 GB)	1 Gbps (full SOHO rate)	∼80% (no throttling)	None at 1 Gbps (0% drop)

4.4. Central Node and Agent Overhead Evaluation

To quantify the resource overhead introduced by the R-Snort stack running on a single node, a dedicated measurement experiment was conducted on the central node. The full stack was measured (Snort, rsnort_agent, MariaDB, Grafana, and ClamAV), with a 120 s idle phase followed by a 300 s load phase, sampling every 1 s via */proc* on interface `eth0`.

Table 3 summarises the results of the test, which consisted of a 120 s idle phase followed by a 300 s load phase, sampled every 1 s. The most significant finding is that the total memory overhead introduced by the active load is minimal: the Proportional Set Size (PSS) increases by only 2.8 MiB between idle and load conditions (514.4 MiB vs. 517.2 MiB). CPU usage remains below 1.1% in both phases, confirming that the central node's management stack imposes negligible computational overhead on the host. Network throughput on the administration interface rises from 0.33 KB/s (idle) to 11.62 KB/s (load), reflecting the alert ingestion traffic from the monitored agent.

Table 3. Central node resource overhead: idle vs. load.

Metric	Idle	Load
Total PSS (MiB)	514.4	517.2
Total RSS (MiB)	1583.6	1586.3
CPU mean (%)	1.02	0.90
CPU p95 (%)	0.90	0.90
RX throughput (KB/s)	0.33	11.62
TX throughput (KB/s)	0.20	0.18
Alert rate (alerts/min)	—	16.05

Table 4 provides a per-component breakdown of memory (mean PSS). Grafana accounts for the largest share (254.5 MiB), followed by MariaDB (115.8 MiB) and Snort (88.9–91.7 MiB). The rsnort_agent process itself consumes approximately 55.3 MiB and exhibits no measurable increase under load, confirming its lightweight design.

These results validate that R-Snort's full management stack operates well within the memory capacity of the Raspberry Pi 5 (8 GB), leaving substantial headroom for additional agents and correlated traffic. The alert generation rate observed under load (16.05 alerts/min) demonstrates active detection operation during the measurement.

Figure 11 shows the R-Snort agent memory (PSS and RSS) over time, confirming stable behaviour across both phases with no memory leaks or growth under load. Figure 12 shows per-component CPU usage over time; the initial spike visible at $t = 0$ corresponds to the startup initialisation of the rsnort_agent process (up to 11.8%), after which all components settle below 0.5%.

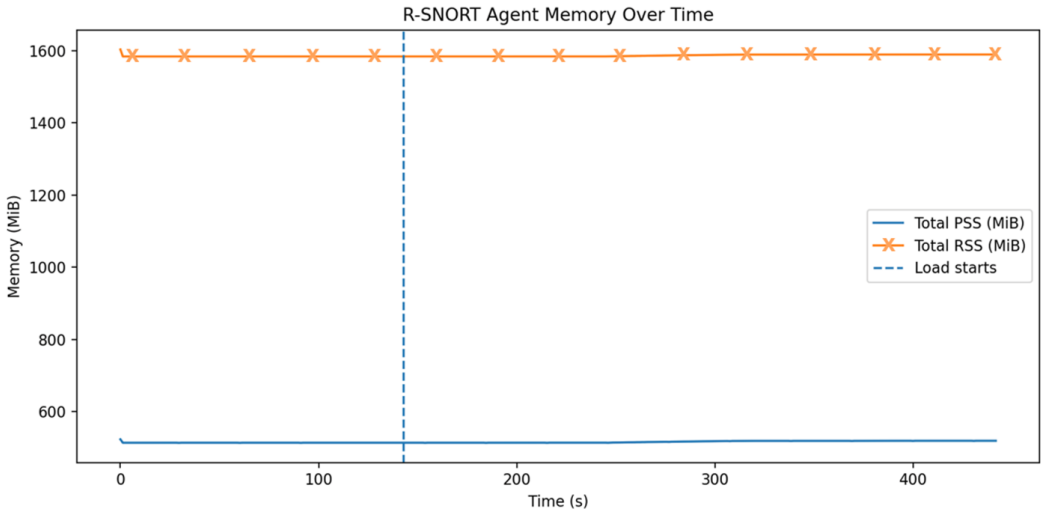


Figure 11. R-Snort agent memory over time (PSS and RSS, in MiB).

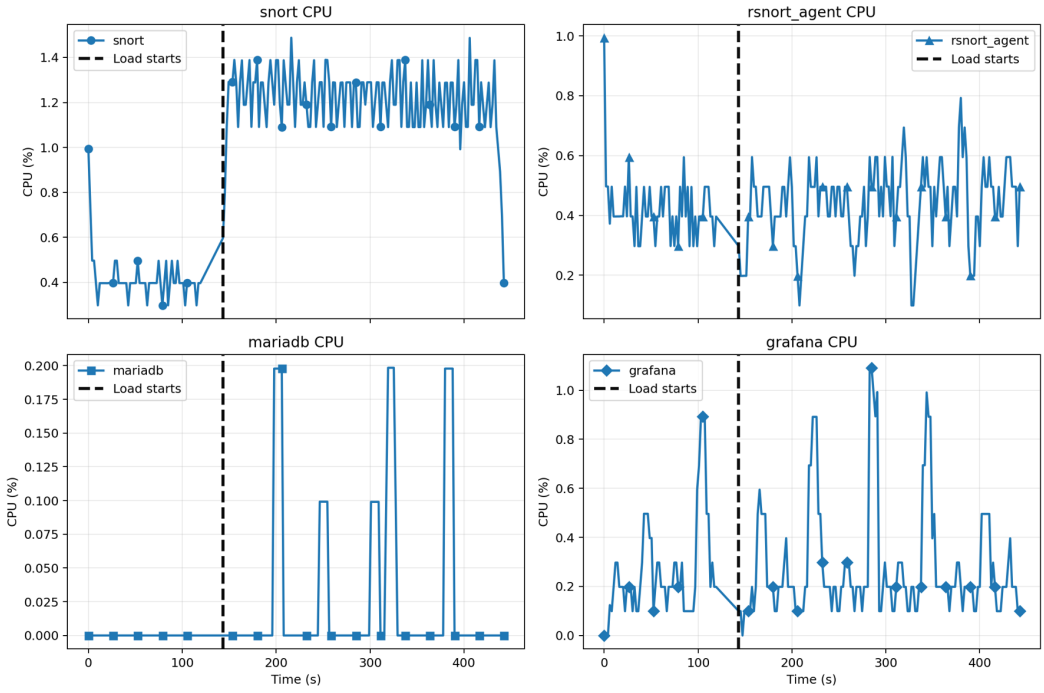


Figure 12. Per-component CPU usage over time (%).

Figure 13 shows the network throughput on the administration interface over time. During the idle phase, RX traffic remains negligible, rising to approximately 19 KB/s once the load phase begins, reflecting alert ingestion traffic from the monitored agent. TX throughput remains near zero throughout both phases, confirming that the central node's outbound traffic is minimal.

Table 4. Per-component mean PSS memory (MiB): idle vs. load.

Component	Idle (MiB)	Load (MiB)
snort	88.87	91.68
rsnort_agent	55.31	55.24
mariadb	115.72	115.80
grafana	254.49	254.50
Total	514.39	517.22

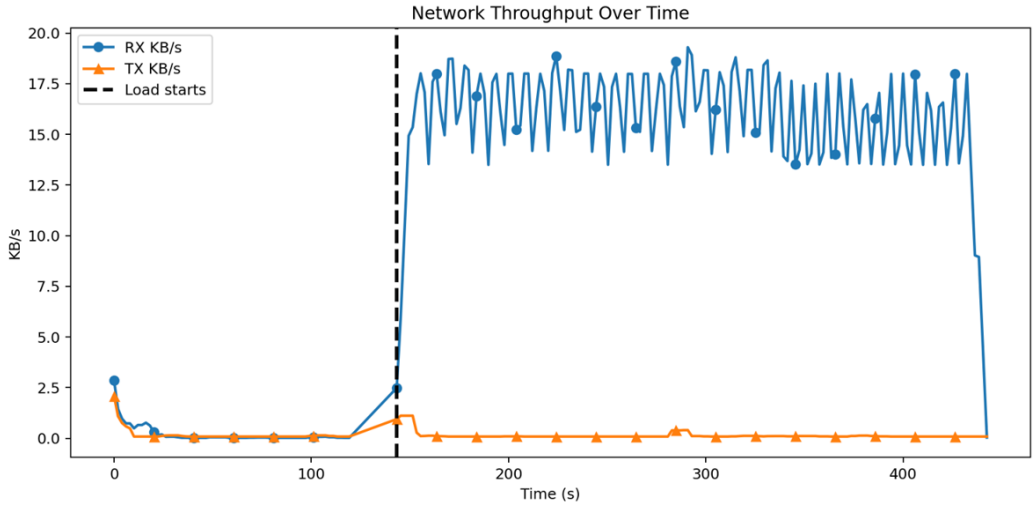


Figure 13. Network throughput over time (KB/s).

5. Discussion

The results obtained from the functional and performance tests validate the technical and operational viability of R-Snort as an effective NIDS for SOHO and EoT networks. In line with the initial objectives, the system offers advanced detection capabilities without compromising the stability of the environment or requiring specialized hardware.

In terms of resources, it is confirmed that the impact of Snort 3 on the system is moderate under normal conditions, and that, under high-traffic loads, while CPU and RAM usage increases, the system remains stable. Section 4.3 contextualises these results against the performance figures documented in prior literature for Raspberry Pi-based Snort deployments [9,26], where packet loss was observed at traffic rates orders of magnitude below the 1 Gbps achieved by R-Snort. Furthermore, Section 4.4 provides a quantitative characterisation of the central node overhead: the full R-Snort stack (Snort, rsnort_agent, MariaDB, Grafana, and ClamAV) consumes 514.4 MiB PSS at idle and increases by only 2.8 MiB under load, with CPU usage remaining below 1.1% in both phases. This confirms that the management stack imposes negligible overhead on the host hardware. This tolerance validates that the modular design and the use of the Raspberry Pi 5 are suitable for SOHO and Edge-of-Things (EoT) networks.

Among the main strengths of R-Snort is its cost-effectiveness, as it allows the deployment of a complete monitoring system with low-cost hardware and free software. Its modular architecture and automated installer facilitate rapid deployment, even for users without advanced technical knowledge. Integration with tools such as Grafana and MariaDB provides a clear and professional visualization of alerts, improving the user experience, and facilitating event analysis.

Beyond its cost-effectiveness, the system also demonstrates notable scalability. Although limited by the hardware used, the architecture allows for the easy integration of additional agents, which automatically connect to the central control panel. In addition,

the ability to manage rules from the graphical interface and download logs reinforces its usefulness in forensic or auditing processes.

The implementation of R-Snort has direct implications for applied cybersecurity in SOHO networks. In a context where the most accessible NIDS solutions are aimed at companies with larger resources, R-Snort provides a functional and professional alternative for SOHO administrators, without giving up key features such as signature detection, metrics collection, or real-time alert visualization. In short, R-Snort represents a decisive step towards the democratization of cybersecurity, allowing small organizations, communities, or individuals to deploy their own NIDS with reduced effort and cost.

The current evaluation is subject to several limitations. First, the test environment comprised only three agents in a local LAN for the multi-agent tests (Section 4.2), and while Section 4.4 provides a single-node overhead characterisation, a full multi-node scaling experiment measuring central node overhead as a function of the number of connected agents remains as future work. Second, the experiments focused primarily on performance metrics; a comprehensive detection accuracy evaluation using a labeled attack dataset would further strengthen the scientific contribution. Third, complex evasion techniques against the detection engine were not evaluated.

6. Conclusions

Unlike previous studies focused on the Raspberry Pi 4, which identified significant bottlenecks when evaluating NIDS performance under real-world network conditions [9,26], R-Snort uses the Raspberry Pi 5 architecture to achieve enterprise-level efficiency on low-cost hardware.

R-Snort is a robust and versatile tool that allows the detection of network attacks. The intuitive web interface simplifies the configuration and visualization of alerts, democratizing access to advanced cybersecurity for users without extensive knowledge.

One of R-Snort's key innovations is its multi-agent architecture. Unlike a single-sensor system, this configuration allows the centralized module to correlate alerts from multiple agents, identifying distributed attack patterns. Tests have shown that R-Snort is capable of detecting coordinated attacks with multiple origins, which validates the system's effectiveness in providing complete network visibility.

The development of R-Snort demonstrates that it is possible to implement a comprehensive and multi-agent intrusion detection system in SOHO and EoT networks. The project meets its main objectives, as it offers an advanced and easy-to-use cybersecurity solution, capable of operating stably and efficiently on a Raspberry Pi 5. Performance tests validate that the impact on the CPU and RAM is acceptable, even under stress tests. The central node overhead evaluation in Section 4.4 further confirms that the full management stack operates within 517 MiB PSS and below 1.1% CPU in both idle and load conditions, leaving substantial hardware headroom. The architectural advances of the RPi5—specifically the Cortex-A76 CPU and the RP1 southbridge for I/O offloading—are the key enablers of 1 Gbps zero-packet-loss operation demonstrated in Section 4.1. These results confirm the technical and operational viability of the solution.

Future work will focus on three directions: (1) evolving R-Snort into a full Intrusion Prevention System (IPS) with real-time automated blocking of malicious hosts; (2) incorporating anomaly-based and machine learning-based detection to complement the current signature-based approach; and (3) validating the architecture in larger-scale deployments with more agents, WAN latency conditions, and a formal detection accuracy evaluation using standardized attack datasets. A formal security analysis of the agent communication architecture against adversarial attacks is also identified as a critical future direction. These

improvements will allow R-Snort to evolve into a proactive security platform capable of detecting, correlating, and responding to threats more efficiently.

Author Contributions: J.G.L. designed the overall system architecture, led the final system improvements of the software, and supervised the research. D.O.P.T. contributed to the system design and was responsible for the core software implementation and development. N.P.S. and A.A.G. performed the data analysis and contributed to the experimental setup, network deployment, and performance evaluation. All authors have read and agreed to the published version of the manuscript.

Funding: This research was funded by the research groups "Informática Aplicada (TIC211)" and "Computación, Optimización y Sensorización en Ingeniería y Energía (TIC221)".

Data Availability Statement: The source code, configurations, and installation scripts for the R-Snort multi-agent system are publicly available in the following GitHub repository: https://github.com/jugomezual/R-Snort (accessed on 29 March 2026). This repository includes the custom Snort 3 compilation setup, the multi-agent architecture, and the necessary supporting documentation to reproduce the findings and deploy the system described in this study.

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

References

1. Ang, S.; Ho, M.; Huy, S.; Janarthanan, M. Utilizing IDS and IPS to Improve Cybersecurity Monitoring Process. *J. Cyber Secur. Risk Audit.* **2025**, *2025*, 77–88. [CrossRef]
2. Adeleke, O. Intrusion Detection: Issues, Problems and Solutions. In Proceedings of the 3rd International Conference on Information and Computer Technologies (ICICT), San Jose, CA, USA, 9–12 March 2020; pp. 397–402. [CrossRef]
3. Snort. Snort—Network Intrusion Detection and Prevention System. 2025. Available online: https://www.snort.org/ (accessed on 29 March 2026).
4. Waleed, A.; Jamali, A.; Masood, A. Which open-source IDS? Snort, Suricata or Zeek. *Comput. Netw.* **2022**, *213*, 109116. [CrossRef]
5. Hu, Q.; Yu, S.Y.; Asghar, M.R. Analysing performance issues of open-source intrusion detection systems in high speed networks. *J. Inf. Secur. Appl.* **2020**, *51*, 102426. [CrossRef]
6. Karim, I.; Vien, Q.-T.; Le, T.A.; Mapp, G. A Comparative Experimental Design and Performance Analysis of Snort-Based Intrusion Detection System in Practical Computer Networks. *Computers* **2017**, *6*, 6. [CrossRef]
7. Gómez, J.; Gil, C.; Padilla, N.; Baños, R.; Jiménez, C. Design of a Snort-Based Hybrid Intrusion Detection System. In *Distributed Computing, Artificial Intelligence, Bioinformatics, Soft Computing, and Ambient Assisted Living. IWANN 2009*; Lecture Notes in Computer Science; Springer: Berlin/Heidelberg, Germany, 2009; Volume 5518. [CrossRef]
8. Boukebous, A.A.E.; Fettache, M.I.; Bendiab, G.; Shiaeles, S. A Comparative Analysis of Snort 3 and Suricata. In Proceedings of the IEEE IAS Global Conference on Emerging Technologies (GlobConET), London, UK, 19–21 May 2023; pp. 1–6. [CrossRef]
9. Varma, P.; Siddiqui, A.; Vadher, P.; Baloda, V. Snort IDPS using raspberry Pi 4. *Int. J. Eng. Res. Technol. (IJERT)* **2020**, *9*, 151–154.
10. Patel, N.D.; Mehtre, B.M.; Wankar, R. A Snort-based secure edge router for smart home. *Int. J. Sens. Netw.* **2023**, *41*, 42–59. [CrossRef]
11. Sosinski, R. A Single Board Computer Based Solution for Smart Home Network Security. Master's Dissertation, National College of Ireland, Dublin, Ireland, 2025. Available online: https://norma.ncirl.ie/8376/ (accessed on 29 March 2026).
12. Asomami-Wiafe, A.S. Building a Cheap Security Solution for a Smart Home Network Using a Raspberry Pi and Snort 3. Bachelor's Thesis, Ashesi University, Berekuso, Ghana, 2020. Available online: https://air.ashesi.edu.gh/items/0571895b-e4f8-423b-b925-c5ac521eef1b (accessed on 29 March 2026).
13. Coşar, M.; Karasartova, S. A firewall application on SOHO networks with Raspberry Pi and Snort. In Proceedings of the 2017 International Conference on Computer Science and Engineering (UBMK), Antalya, Turkey, 5–8 October 2017; IEEE: New York, NY, USA, 2017; pp. 1000–1003. [CrossRef]
14. Makopa, J.; Christopher, A.; Shah, R.; Mandela, N. Internet of Things (IoT) network forensic analysis using the Raspberry Pi 4 Model B and open source tools. In Proceedings of the 2023 International Conference on Quantum, Technologies, Communications, Computing, Hardware and Embedded Systems Security (QCCHESS), Kottayam, India, 15–16 September 2023; IEEE: New York, NY, USA, 2023. [CrossRef]
15. Saad, S.; Traore, I. Semantic aware attack scenarios reconstruction. *J. Inf. Secur. Appl.* **2013**, *18*, 53–67. [CrossRef]
16. Younes, T.; Jemili, F. A Multi-Agent-Based System for Intrusion Detection. In *Innovations in Smart Cities Applications Volume 4: The Proceedings of the 5th International Conference on Smart Cities Applications*; Springer: Singapore, 2021; pp. 182–195. [CrossRef]

17. Al-Yaseen, W.L.; Othman, Z.A.; Nazri, M.Z.A. Real-Time Intrusion Detection System Using Multi-agent System. *IAENG Int. J. Comput. Sci.* **2016**, *43*, 80–90. Available online: https://www.researchgate.net/publication/299363900_Real-Time_Intrusion_Detection_System_Using_Multi-agent_System (accessed on 19 April 2026).
18. Iyengar, N. Evaluation of Network Based IDS and Deployment of multi-sensor IDS. *arXiv* **2020**. [CrossRef]
19. Carver, C.; Hill, J.; Surdu, J.; Pooch, U. A Methodology for Using Intelligent Agents to provide Automated Intrusion Response. In Proceedings of the IEEE Systems, Man, and Cybernetics Information Assurance and Security Workshop, West Point, NY, USA, 6–7 June 2000.
20. Tu, J.; Wang, T.; Wang, J.; Manivasagam, S.; Ren, M.; Urtasun, R. Adversarial attacks on multi-agent communication. In Proceedings of the IEEE/CVF International Conference on Computer Vision, Montreal, BC, Canada, 11–17 October 2021; pp. 7768–7777.
21. Davies, T.; Hashem, M.; Shone, N.; Lyon, R. A Collaborative Intrusion Detection System Using Snort IDS Nodes. *arXiv* **2025**. [CrossRef]
22. Sakovich, N.; Aksenov, D.; Pleshakova, E.; Gataullin, S. A Neural Operator Using Dynamic Mode Decomposition Analysis to Approximate Partial Differential Equations. 2025. Available online: https://ssrn.com/abstract=5542238 (accessed on 19 April 2026).
23. Elasticsearch, B.V. The Elastic Stack: Elasticsearch, Logstash, and Kibana. 2026. Available online: https://www.elastic.co/elastic-stack (accessed on 29 March 2026).
24. R-Snort Official Repository. Version v1.0. Available online: https://github.com/jugomezual/R-Snort (accessed on 29 March 2026).
25. Bar-Or, T. Snort IDS/IPS Dashboard [Grafana Dashboard]. Grafana Labs. Available online: https://grafana.com/grafana/dashboards/3099 (accessed on 22 February 2026).
26. Sforzin, A.; Gómez Mármol, F.; Conti, M.; Bohli, J.-M. RPiDS: Raspberry Pi IDS—A Fruitful Intrusion Detection System for IoT. In Proceedings of the 2016 Intl IEEE Conferences on Ubiquitous Intelligence & Computing, Advanced and Trusted Computing, Scalable Computing and Communications (UIC/ATC/ScalCom), Toulouse, France, 18–21 July 2016; pp. 440–448. [CrossRef]
27. Raspberry Pi Ltd. Raspberry Pi 5 Product Brief and Processor Documentation: BCM2712 and RP1 Southbridge, Raspberry Pi Documentation. 2025. Available online: https://www.raspberrypi.com/products/raspberry-pi-5/ (accessed on 29 March 2026).

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