DOI: https://doi.org/10.61762/pjstvol8iss2art002

PLATFORM - A Journal of Science & Technology

Received: 22 October 2025, Revised: 22 December 2025, Accepted: 22 December 2025, Published: 31 December 2025, Publisher: UTP Press, Creative Commons: CC BY 4.0

A RASPBERRY PI-BASED FRAMEWORK FOR NETWORK SECURITY THREAT ANALYSIS IN LOCAL AREA NETWORKS

Amirul Hakim Muhamad Hafidz[1], Noureen Talpur[1]*, Norshakirah A Aziz[1], Mohd Hafizul Afifi Abdullah[2]

[1]Department of Computing, Universiti Teknologi PETRONAS, Malaysia
[2]Department of Computer Science, Universiti Tunku Abdul Rahman, Malaysia

*Corresponding author: noureen.talpur@utp.edu.my

ABSTRACT

The growing complexity of local area networks (LANs), along with restricted access to enterprise-grade monitoring systems, has made small-scale networks vulnerable to cyber threats like reconnaissance, denial-of-service (DoS) attacks, and unauthorised access. This paper describes the design and implementation of a lightweight, Raspberry Pi-based network threat analytics system that provides real-time monitoring and anomaly detection in LAN contexts. The system utilises open-source technologies, including Node-RED, Nmap, ifstat, hping3, and Netcat, to provide a multi-tab dashboard that displays active hosts, service availability, latency trends, and bandwidth usage. To simulate a realistic network environment, an isolated LAN was set up with a TP-Link AC1200 router and UMobile Home 5G connectivity. During the testing phase, the Raspberry Pi 400 operated as both a monitoring node and an attack simulator. Three attack scenarios, SYN Flood, TCP Port Scan, and Honeypot attack simulation, were used to test the system's detection and visualisation skills. The system successfully identified volumetric traffic irregularities, reconnaissance activities, and interactions with misleading services while incurring minimum CPU and memory overhead. Evaluations of performance confirmed the system's capacity to run on limited hardware while maintaining good responsiveness across all tabs within a single functional Node-RED dashboard. This study demonstrates the viability of installing low-cost, modular, and scalable network analytics solutions in resource-constrained situations such as home networks, educational labs, and small to medium-sized businesses. The suggested solution adheres to Zero Trust principles by requiring continuous verification of network operations and provides a practical foundation for improving situational awareness and early threat identification.

Keywords: *Raspberry Pi, Node-RED, network analytics, Agile SDLC, SYN flood, lightweight monitoring*

INTRODUCTION

The rapid expansion of networked devices has converted local area networks (LANs) into highly dynamic environments that support a wide range of vital applications. While large companies use advanced security information and event management (SIEM) platforms and intrusion detection systems (IDS) for real-time monitoring, small-scale networks in households, educational institutions, and small businesses frequently lack these safeguards. These environments are nonetheless highly vulnerable to cyber threats, such as reconnaissance, denial-of-service (DoS) attacks, and unauthorised access attempts [1]. High-cost solutions, along with the need for specialised hardware and expertise, make it challenging to implement enterprise-grade monitoring systems in resource-constrained environments. As a result, there is an urgent need for lightweight, cost-effective, and easily deployable solutions that can improve network visibility and situational awareness without requiring significant resource overhead.

Existing LAN security systems are often positioned at two extremes: they are either overly simplistic, offering only basic monitoring with little to no analytical capability, or excessively complex, requiring significant computational resources and advanced expertise to

operate [2]. While enterprise-grade solutions may provide comprehensive protection, they are generally impractical for smaller networks due to high costs, maintenance requirements, and hardware demands. Conversely, lightweight tools tend to lack the ability to perform real-time analytics or provide meaningful visualisations of network behaviour. Although many open-source alternatives exist, their deployment and integration typically demand considerable technical knowledge, limiting accessibility for non-expert users such as small businesses, educational labs, or home networks. This unaddressed gap leaves small-scale networks vulnerable to a wide range of attack vectors, including TCP port scans, SYN Floods, and attempts to exploit deliberately exposed or misconfigured services [3]. In the absence of clear and user-friendly dashboards, malicious activities can persist undetected, thereby increasing the risk of malware propagation, service disruptions, and the compromise of sensitive information.

Hence, the goal of this research is to develop and implement a lightweight network threat analytics system utilising the Raspberry Pi 400 hardware platform. The system utilises open-source tools such as Node-RED, Nmap, ifstat, hping3, and Netcat to create a multi-tab Node-RED dashboard that visualises active hosts and open ports, monitors network health and latency, analyses bandwidth usage patterns, and detects anomalies caused by simulated attacks in real-time. This method demonstrates how small-scale networks can offer meaningful situational awareness without requiring expensive infrastructure.

LITERATURE REVIEW

Small LANs, particularly in residences, academic institutions, and small businesses, often lack the comprehensive monitoring systems found in large-scale networks. Despite their limited visibility, these environments remain vulnerable to reconnaissance, DoS attacks, and unauthorised access [1],[4]. To address these concerns, Osman et al. [1] designed and implemented a TorVPN access point using a Raspberry Pi to enhance LAN security and privacy. Their confidentiality test demonstrated 100% encryption success, effectively preventing password sniffing and Man-in-the-Middle attacks. However, Internet connectivity performance degraded significantly, with average ping ranging from 744.962 ms to 1007.298 ms, download speeds between 2.17 Mbps and 3.28 Mbps, and upload speeds between 2.20 Mbps and 2.94 Mbps across different Tor exit nodes. The study primarily focused on privacy and confidentiality through the integration of Tor and VPN, rather than implementing real-time intrusion detection or anomaly-based monitoring. Although the approach improved anonymity, it lacked scalability analysis, comparative benchmarking, and optimisation for latency-sensitive applications, highlighting a gap in real-time threat detection and performance tuning for practical deployment.

The Raspberry Pi's affordability and versatility have made it a popular platform for lightweight security applications. Zhang et al. [5] implemented an ML-based IoT intrusion detection pipeline using Node-RED. Their initial desktop evaluation achieved ~90% accuracy for general traffic, 99% under similar network conditions, and a 95% attack detection rate using One-Class models (OCSVM, EE, IF, LOF). When deployed on an ISP router (dual-core Atom, 640 MB RAM), the Node-RED pipeline showed an average response time of 4.61 ms (prediction) and 5.60 ms (preprocessing + prediction), confirming its feasibility for edge-based detection.

Tripathi and Kumar [6] proposed an intrusion detection system based on Raspberry Pi, integrating Snort with a custom dashboard. The system achieved a 100% detection rate for all simulated attacks, including ICMP ping and SSH brute-force attempts, while successfully capturing network traffic without any packet loss. No memory or processing errors were observed on the Raspberry Pi 3, demonstrating its feasibility as a low-cost security solution. However, the system did not support continuous host discovery using tools such as Nmap.

De Voegleer et al. [7] concluded that a network cannot be 100% secure, as shown through simulation, where default configurations allow unauthorised access, whereas overly restrictive policies can entirely disrupt normal network operations. Their findings emphasise that effective security relies on proper configuration and thorough network analysis rather than reliance on individual security products. In related work, the authors also proposed a comprehensive LAN security system integrating packet sniffing and real-

time monitoring; however, its complexity and high resource requirements limit its suitability for resource-constrained environments.

Node-RED, an open-source flow-based programming tool, has gained traction as a lightweight solution for data processing and visualisation [8]. This study of [9] investigates the synergy between Node-RED and IoT analytics to address the complexities of real-time data processing. The authors demonstrate that by utilising Node-RED's visual programming interface alongside real-time analytical tools, developers can significantly reduce the technical overhead and time required to build IoT applications. A practical case study of an industrial IoT system showcases the framework's ability to aggregate data from multiple sensors into a flexible and scalable visualisation platform. Additional tools, such as Ifstat, provide lightweight bandwidth statistics for real-time traffic monitoring. Meanwhile, Hping3 and Netcat are widely used in academia to simulate SYN Flood attacks and deploy honeypots [10].

This study addresses the identified gaps by proposing a Raspberry Pi-powered network threat analytics system that integrates Nmap, Ifstat, Hping3, and Netcat within a multi-tab Node-RED dashboard. The system enables small-scale networks to detect SYN Flood attacks, port scans, and honeypot interactions in real time, demonstrating a practical balance between functionality and lightweight performance on constrained hardware.

METHODS

The project employed the Agile software development life cycle (SDLC) (Figure 1) due to its incremental and iterative methodology, which aligns well with the research project's exploratory and experimental design [11]. Agile approaches, as opposed to traditional waterfall models, enable flexible adaptation to changing needs and problems as the project is being implemented.

The Agile model divides the project into multiple sprints (short cycles), enabling functional components to be tested and validated early. This approach suits this project due to its emphasis on rapid prototyping (e.g., Node-RED dashboards), validation (through attack simulations), and continuous improvement.

The proposed system architecture, as illustrated in Figure 2, is a lightweight, modular network monitoring solution for small-scale and resource-constrained LAN environments. It comprises a TP-Link AC1200 router, which provides both wired and wireless connectivity and acts as the primary LAN distribution point, and a Raspberry Pi 400 that serves as both the central monitoring server and the attack simulation node. The Raspberry Pi hosts Node-RED to collect, analyse, and visualise LAN traffic, perform host and port discovery via Nmap, monitor latency and bandwidth, and aggregate metrics in a multi-tab dashboard. It also simulates common network attacks, including SYN Flood, TCP port scans, and data exfiltration, to validate the system's detection capability.

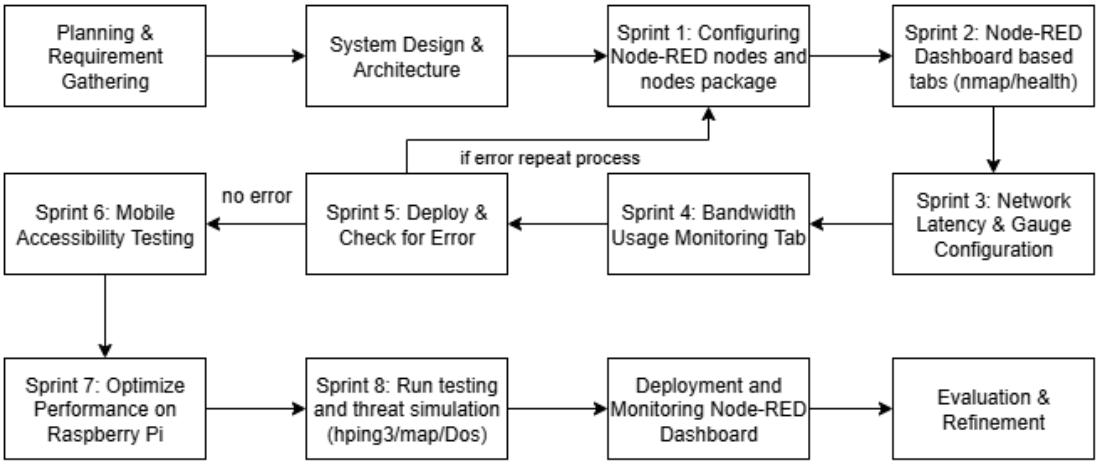


Figure 1 Agile SDLC sprint diagram

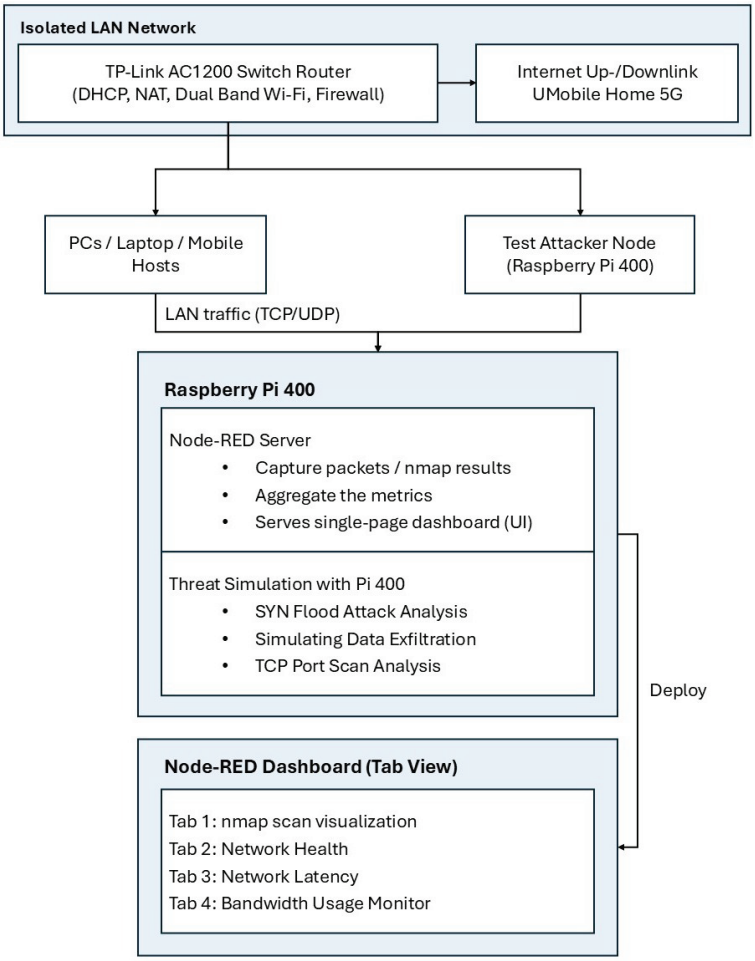


Figure 2 System design flow

Moreover, the Node-RED dashboard presents real-time insights across four tabs demonstrating how low-cost hardware can effectively support comprehensive LAN threat analytics in an isolated environment. The overview of each tab is given as follows:

1. Tab 1: Nmap Scan Visualisation performed periodic subnet scanning to display active hosts, open ports, and associated services through the automatic execution of Nmap commands.
2. Tab 2: The Network Health tracked uptime, service availability, and failures by sending ICMP ping requests to designated targets.
3. Tab 3: The Network Latency monitors round-trip times (RTT) to critical servers, providing insight into link quality.
4. Tab 4: The Bandwidth Usage page used ifstat to monitor upload and download speeds over time, emphasising any unexpected spikes that could indicate data exfiltration or DoS activity.

To validate the system's capability to detect anomalies, three attack scenarios were simulated within the isolated LAN. A SYN Flood attack was generated using the hping3 tool, where a large volume of TCP SYN packets was directed at port 80 of a target host to overwhelm its resources. This simulation tested the system's ability to identify abnormal traffic patterns and service unavailability. A TCP Port Scan was conducted using Nmap's stealth scan option (nmap -sS -T4 192.168.1.0/24) to emulate reconnaissance activity commonly used by attackers to discover open ports and services. The final scenario involved honeypot probing, where Netcat was deployed to emulate a fake HTTP service on port 8080, allowing the system to detect and log unauthorised connection attempts.

Overall, this methodology ensures a comprehensive evaluation of the proposed system in realistic LAN conditions. It demonstrates how open-source tools can be orchestrated within a Raspberry Pi-based platform to provide continuous visibility and early threat detection

without the complexity and cost of enterprise-grade solutions. The detailed implementation workflow reflects a lightweight yet robust approach, making it suitable for deployment in resource-constrained environments, such as home networks, educational institutions, and small enterprises.

RESULTS AND DISCUSSION

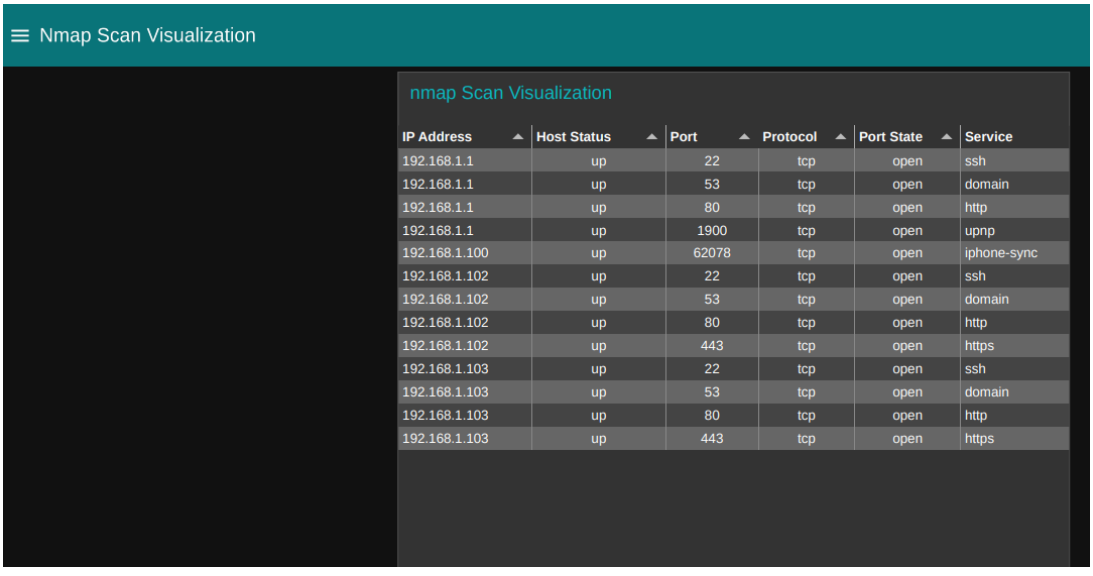
As mentioned in the previous section, the Node-RED dashboard is created with four primary tabs, each covering a different component of network analytics. The "Nmap Scan Visualisation tab" displays a real-time map of active devices on the LAN, as well as open ports and services discovered during periodic Nmap scans, providing information about the current network footprint and potential vulnerabilities. The "Network Health tab" utilises real ping data to determine device status (up or down) and detects downtime or unreachability. The "Network Latency tab" displays round-trip latency for essential endpoints, such as OpenDNS (4.2.2.2) and Google DNS (8.8.8.8), as well as internal LAN devices. Gauge widgets and trend graphs provide a clear perspective on response time performance over time. Finally, the "Bandwidth Usage Monitor tab" tracks upload and download rates (in Mbps), providing real-time and historical bandwidth trends to identify anomalous traffic spikes, which is important for detecting excessive data transfers that may indicate data exfiltration attempts. The details of each tab and accumulated results are explained in the following subsections.

Tab 1: Nmap Scan Visualisation

The Node-RED dashboard in Figure 3 shows the first tab of the Nmap scan visualisation. This tab displays active devices and open ports in real time within the isolated LAN environment. This functionality utilises Nmap, a prominent open-source network scanning tool, which is integrated into Node-RED via an exec node to execute automated scans.

Figure 4 illustrates the Node-RED flow for Tab 1, which is composed of five core nodes as follows:

1. "Start Nmap Scan" (Inject Node) command initiates the scanning process, which can be done manually or at predetermined intervals.
2. "Run Nmap" (Exec Node) runs the command nmap -oX - 192.168.1.102/24, which scans all devices on the 192.168.1.0/24 subnet and outputs the data in XML format for later analysis.
3. "Parse Nmap XML" (XML Node) parses the XML output and extracts all pertinent data.
4. "Format Scan Data" (Function Node) next processes the parsed data, converting it into a JSON object suitable for dashboard visualisation.
5. The "Nmap Scan Results" (UI Table Node) displays the processed data in a clear, tabular style on the dashboard.



IP Address	Host Status	Port	Protocol	Port State	Service
192.168.1.1	up	22	tcp	open	ssh
192.168.1.1	up	53	tcp	open	domain
192.168.1.1	up	80	tcp	open	http
192.168.1.1	up	1900	tcp	open	upnp
192.168.1.100	up	62078	tcp	open	iphone-sync
192.168.1.102	up	22	tcp	open	ssh
192.168.1.102	up	53	tcp	open	domain
192.168.1.102	up	80	tcp	open	http
192.168.1.102	up	443	tcp	open	https
192.168.1.103	up	22	tcp	open	ssh
192.168.1.103	up	53	tcp	open	domain
192.168.1.103	up	80	tcp	open	http
192.168.1.103	up	443	tcp	open	https

Figure 3 Tab 1 showing Nmap scan visualisation

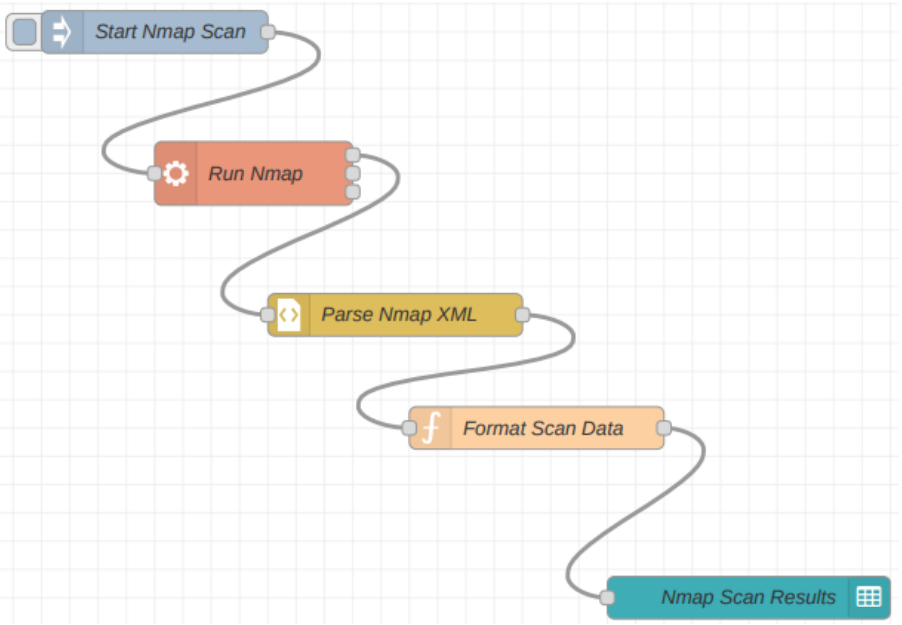


Figure 4 Node-RED nodes configurations in Tab 1

This procedure enables users to perform network scans and visualise the current LAN topology, including IP addresses, host status, open ports, protocols, and services. Using Node-RED's exec node, an Nmap scan (nmap -oX - 192.168.1.102/24) of the 192.168.1.0/24 subnet identified six active devices, including a TP-Link router, Raspberry Pi, PC, and iPhone, with services such as SSH, HTTP, and HTTPS accessible. The dashboard confirms expected devices are operational while highlighting potential security risks, such as UPnP on the router and Apple device synchronisation services. By providing real-time visibility, the system can detect unauthorised hosts, rogue services, and unusual port activity during normal operation or threat simulations. This aligns with the Zero Trust principle, which ensures continuous monitoring to prevent any device or service from being implicitly trusted, thereby supporting microsegmentation, least privilege access, and dynamic security policy adaptation. Table 1 presents the observations of active devices on the LAN network, including their IP addresses and device types.

Table 1 Observation of active devices on the LAN network

IP Address	Device Description
192.168.1.1	TP-Link AC1200 Router
192.168.1.100	Likely a Mobile Host (iPhone)
192.168.1.102	Raspberry Pi 400 (Monitor)
192.168.1.103	LAN Client Device (PC/Laptop)

Tab 2: Network Health Visualisation

The network health analytics dashboard in Figure 4 is configured to monitor the live connectivity and stability of key network endpoints. The flow architecture was developed to deliver ICMP ping requests to primary and secondary hosts within the isolated LAN and process the results to assess connection dependability. The deployment process began by configuring the inject node to have a ping interval of 10 seconds, ensuring near-real-time monitoring of endpoint availability. The init change node was set up to initialise flow variables such as total_fails, error_flag1, and error_flag2, which served as counters and flags for network health events. The advanced ping node sent ICMP requests to several hosts, while function nodes such as process main ping and process secondary pings analysed the results and updated statuses.

The display results function node was essential for collecting and formatting outputs. It applied a timestamp conversion technique to conditionally update the dashboard widgets for Current Status, Total Failures, and Fail History. Fail events were logged with their start time and duration, and then added to a history array in the flow context. Upon implementation, the dashboard provided a clear visual representation of the network status, as shown in Figure 6 of Tab 2 Visualisation.

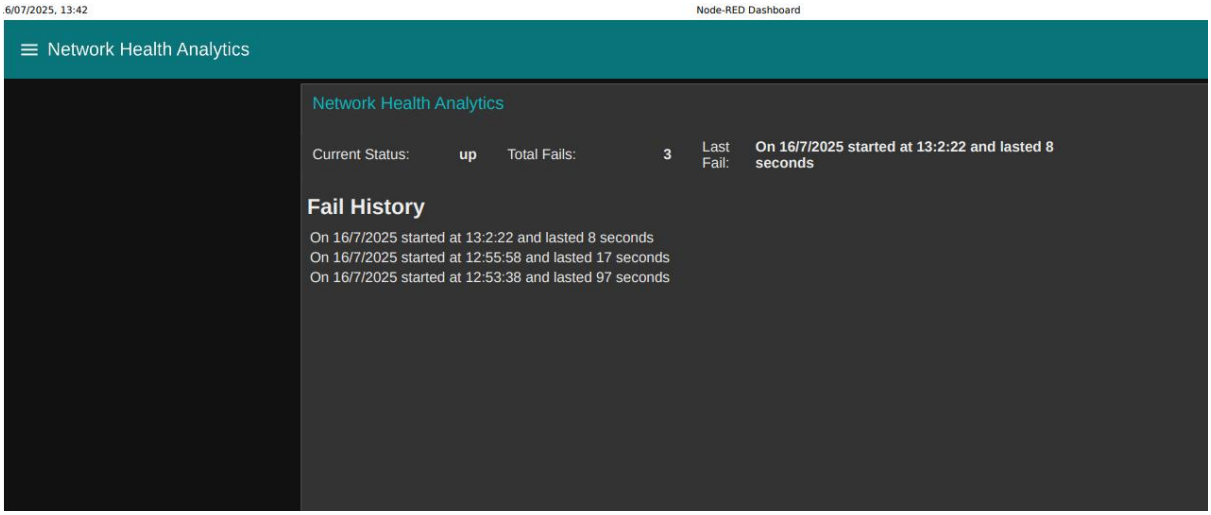


Figure 5 Tab 2 showing network health visualisation

1. Current status: When all monitored hosts responded to pings, the status was displayed as "up".
2. Total Failures: Three interruptions were recorded within the monitoring window.
3. Fail History: Includes specific timestamps and durations for each observed outage (e.g., 97 seconds at 12:53:38).

The JavaScript function script configured on the 'Display Results' node is central to Tab 2 (Network Health Analytics) in Node-RED (Figure 6). It tracks network health by monitoring ping responses, while recording and displaying uptime, downtime, and failure history. Its primary purpose is to format timestamps for human-readable output and to retain only the 10 most recent failure events, thereby preventing dashboard clutter.

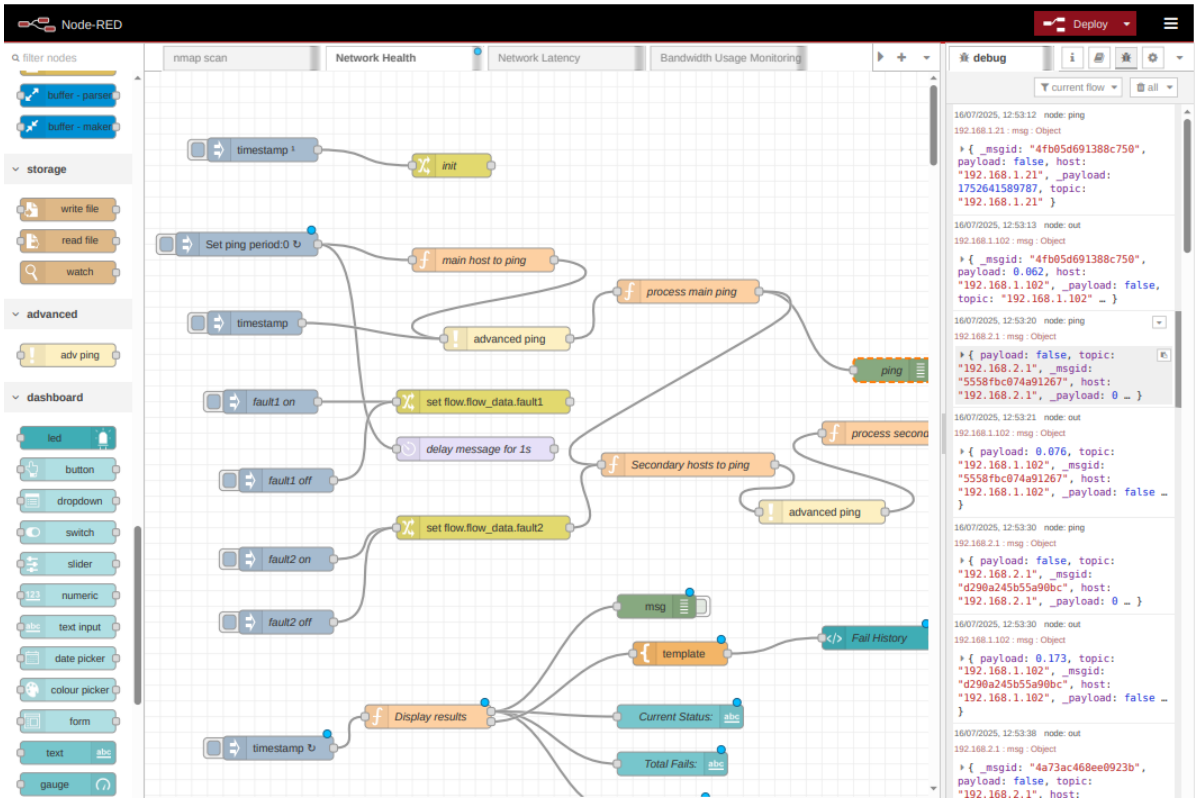


Figure 6 Tab 2 showing Node-RED nodes configurations

1. Retrieve stored data or initialise an empty list: Initially, the function synchronises with the flow context to retrieve the persistent failure history; an empty list is instantiated if no records are found. The process then acquires the live state data comprising status indicators and failure metrics to serve as the basis for subsequent health analytics.

```
var store = flow.get("store")||[];
var flow_data = flow.get("flow_data");
```

2. Get current time and duration: In the second stage, the system instantiates a new Date object to synchronise with the current system time. It concurrently extracts the existing downtime duration from the session data and records the current timestamp in milliseconds to facilitate high-precision temporal calculations.

```
    var d = new Date();
    var duration = flow_data.duration;
    var now = d.getTime();
```

3. Helper function – Format timestamp: The algorithm employs a formatting procedure to decompose raw millisecond timestamps into granular temporal components, including hours, minutes, seconds, and calendar dates. The resulting strings, ret_hours and ret_day, standardise the output format for the user interface, facilitating a clear chronological audit trail of network events.

```
function convertime(timestamp)
{
   var dateObj = new Date(timestamp);
   var date = dateObj.getDate();
   var month = dateObj.getMonth() + 1;
   var year = dateObj.getFullYear();
   var hours = dateObj.getHours();
   var minutes = dateObj.getMinutes();
   var seconds = dateObj.getSeconds();
   var ret_hours = hours + ":" + minutes + ":" + seconds;
   var ret_day = date + "/" + month + "/" + year;

   return [ret_hours, ret_day];
}
```

4. Check the network status: The connectivity status is verified via the flow_data.status parameter. If confirmed as active, the status is converted from a boolean type to a string-based 'up' status, preparing the message payload for real-time visualisation on the network health analytics tab."

```
var status = flow_data.status;
if (status)
{
   status = "up";
   }
```

5. Handle when the network just came back up: For every recovery event, the algorithm determines the failure duration and formats the start time into a standardised summary string (e.g., 'On 16/7/2025 started at 13:02:22 and lasted 8 seconds').

```
if (flow_data.change)
{
      duration = now - flow_data.startTime;
      var ret = convertime(flow_data.startTime);
      var start_time = ret[0];
      duration = "On " + ret[1] + " started at " + start_time + "
      and lasted " + parseInt(duration / 1000) + "
seconds";
```

6. Update history and flow data: The system synchronises the state metadata by clearing the change flag and resetting the startTime variable. Subsequently, the generated failure record is appended to the store array, ensuring the persistent failure history is updated before the data is saved back to the flow context.

```
      flow_data.change = false;
      flow_data.startTime = 0; // reset
      flow_data.duration = duration;
      store.push(duration);
```

7. Keep history limited and handle if the network is down: Historical records are limited to the ten most recent failure events, utilising a truncation logic. Additionally, the system handles persistent downtime by assigning a 'down' status whenever the host remains unreachable.

```
If Length(store) ≥ 10 Then
    Remove oldest entry from $store$ (Shift operation)
End If
Else (from Step 4 status check)
    Set status ← "down"
```

8. Update and save to flow context: The function updates the primary message object (msg) with the current network status, the duration of the most recent failure, and the cumulative number of failure events recorded. It then persists both the updated session data (flow_data) and the failure history (store) back into the Node-RED flow context to ensure continuity in tracking.

```
msg.status = status;
msg.duration = duration;
msg.total_fails = flow_data.total_fails;
flow.set("flow_data", flow_data);
flow.set("store", store);
```

9. Prepare a second message for the fail history and return two outputs. The function constructs a secondary message object (msg1) dedicated to the Fail History widget. To ensure that the most recent failure event appears at the top of the list, the historical log (store) is reversed before assignment. The primary message (msg) is intended for widgets displaying Current Status, Total Failures, and Duration, while the secondary message (msg1) provides the ordered failure history. Finally, both messages are returned as outputs from the Function node. During validation, the debug pane confirmed correct detection of host availability transitions and fault conditions by alternating true and false payloads.

```
let msg1 = {};
msg1.payload = store.slice().reverse();
return [msg, msg1];
```

Tab 3: Network Latency Visualisation

The Network Latency Visualisation tab displays real-time information about the performance and responsiveness of LAN and external network services. This dashboard tab detects anomalies, such as traffic congestion, DoS attacks, or performance degradation caused by misconfigurations or malware activity, by continuously monitoring latency data for critical endpoints.

The tab in Figure 7 was built using Node-RED flows that run periodic ICMP ping operations on internal and external hosts, process the response times, and display them in an intuitive dashboard with live server status indicators and gauge-type visualisations after careful deployment. This functionality aligns with the project's goal of delivering lightweight and proactive network threat analytics on the Raspberry Pi OS.

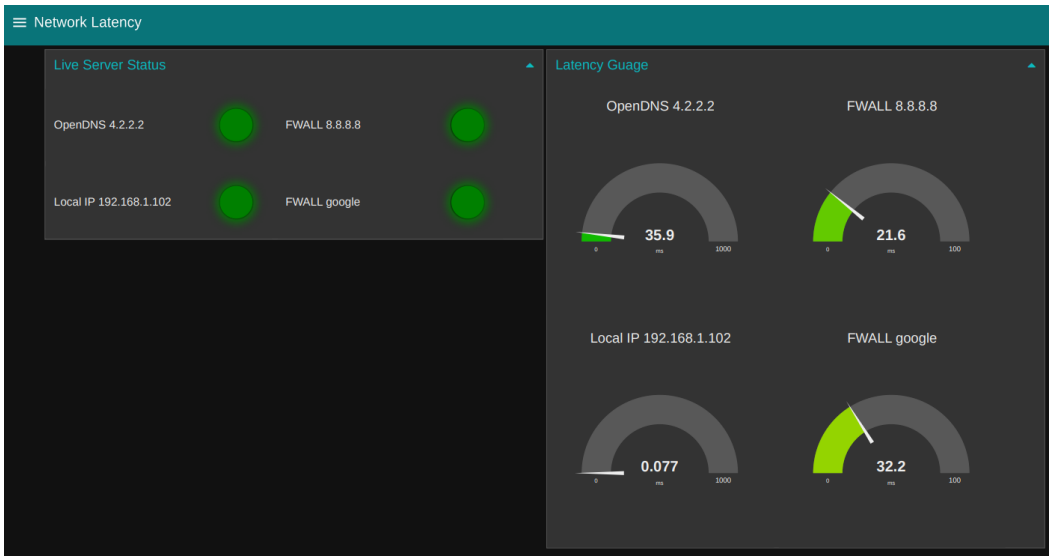


Figure 7 Tab 3 of network latency visualisation

The implementation of Tab 3 followed the flow depicted in Figure 8. The design includes the following:

1. Ping Nodes: Send ICMP echo queries to the specified endpoints and collect round-trip time (RTT) information.
2. Function Nodes: Process raw latency values and handle potential exceptions, such as timeouts and packet loss. The routines also prepare the results for presentation on the dashboard.
3. UI Gauge Widgets: Visualise RTT measurements in milliseconds to quickly assess latency patterns.
4. UI LED Indicators: Display a binary up/down status for each endpoint based on ping answers.

The green LED indications visibly confirmed the ongoing reachability of all monitored endpoints, and most importantly, no packet loss or substantial changes were recorded during this time, suggesting a healthy and stable network environment. This created baseline acts as a critical performance reference point, allowing for the detection of any abnormal behaviour during subsequent threat simulations. Besides, Table 2 presents the baseline latency observations across different endpoints.

Results in Table 2 indicate stable DNS responses, with Google DNS showing slightly lower latency than

Table 2 Baseline observation of result

Endpoint	Average Latency (ms)	Observations
OpenDNS 4.2.2.2	35.9	Stable external DNS response
Google DNS 8.8.8.8	21.6	Slightly faster than OpenDNS
Local IP 192.168.1.102	0.077	Negligible latency (internal)
Firewall Google	32.2	Comparable to OpenDNS performance

OpenDNS, while the local IP recorded negligible delay. The firewall's performance was consistent with external DNS services, confirming a stable and healthy network environment.

Tab 4: Bandwidth Usage Monitoring

The Network Bandwidth Visualisation tab was added to monitor real-time upload and download rates in the isolated LAN scenario. It allows a continuous assessment of bandwidth consumption by displaying both current values and historical trends. This tab supplements the other analytics by detecting aberrant traffic patterns such as huge file transfers, data exfiltration, and volumetric attacks. This functionality,

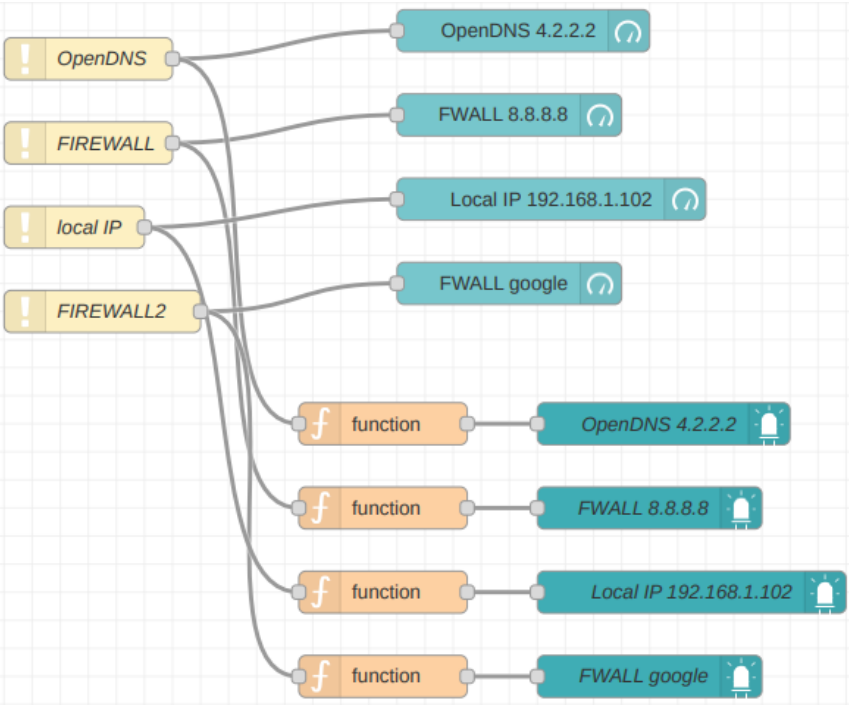


Figure 8 Tab 3 network latency nodes configuration

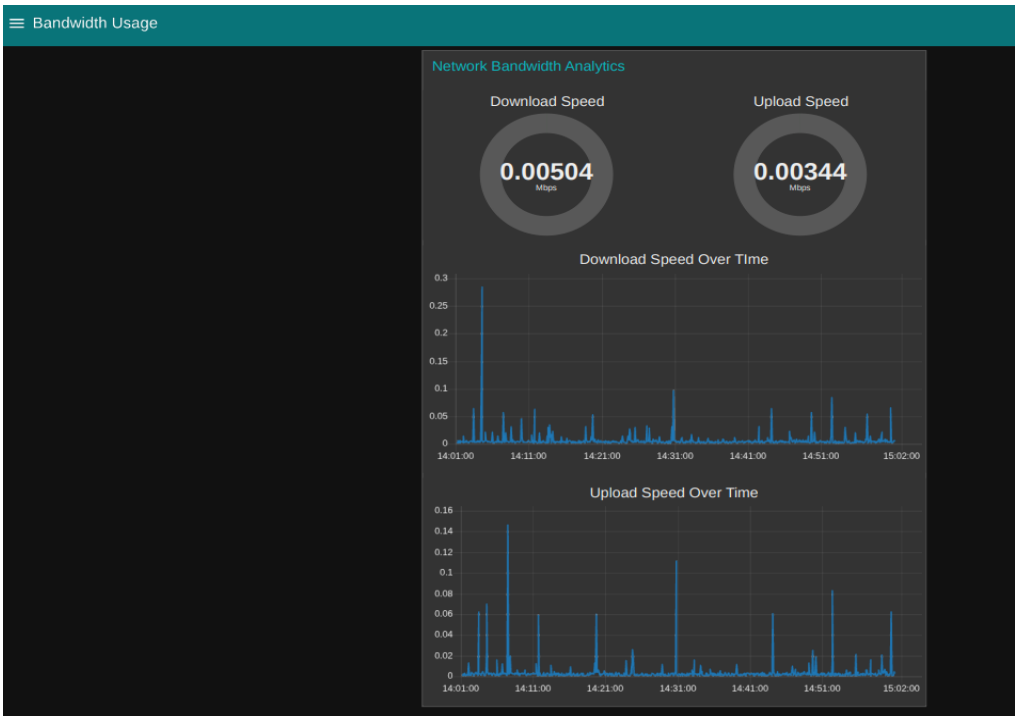


Figure 9 Tab 4 of bandwidth usage monitoring

which runs on the Raspberry Pi 400, utilises lightweight Linux utilities and Node-RED nodes to minimise system overhead while providing meaningful insights into network performance.

Figure 9, which displays Tab 4, provides both real-time and historical bandwidth data, offering useful insights into traffic trends within the monitored LAN. The interface features two doughnut-type gauge charts for instant speeds and two-line graphs for historical trends, providing a thorough understanding of network activities. The doughnut charts prominently display current download and upload speeds in megabits per second (Mbps). At the observed snapshot (in Figure 9):

1. Download Speed: 0.00504 Mbps (~5.04 kbps)
2. Upload Speed: 0.00344 Mbps (~3.44 kbps)

These numbers suggest a low-traffic network state, similar to that of an idle environment with no large-scale transfers or streaming activities. The responsiveness of these charts enables operators to identify spikes immediately, which are reflected by larger full arcs. The doughnut design allows for quick, intuitive interpretation. Low values remain around the chart's centre; however, prolonged traffic loads or bandwidth-intensive activities will then cause the ring to fill proportionally, attracting immediate visual attention on the tab.

Download Speed Over Time Graphs

This graph represents download activity over one hour. Baseline traffic remains consistently low, ranging from 0 to 0.05 Mbps, with occasional spikes reaching 0.3 Mbps. These modest surges are likely due to periodic system updates, DNS queries, or brief HTTP connections from idle clients.

Upload Speed Over Time Graphs

The upload traffic averages less than 0.02 Mbps, with irregular bursts of up to 0.16 Mbps. These bursts could be tiny outgoing packets, such as ARP broadcasts, device health reports, or LAN service ads. Both plots show little evidence of continuous high bandwidth demand, indicating a distinct baseline pattern. This is critical for future comparisons when threat simulations (such as data exfiltration via SCP generate unusual persistent upload activity.

The flow for the tab in Figure 10 consists of four major components:

1. Inject Node (Every 5 Seconds):

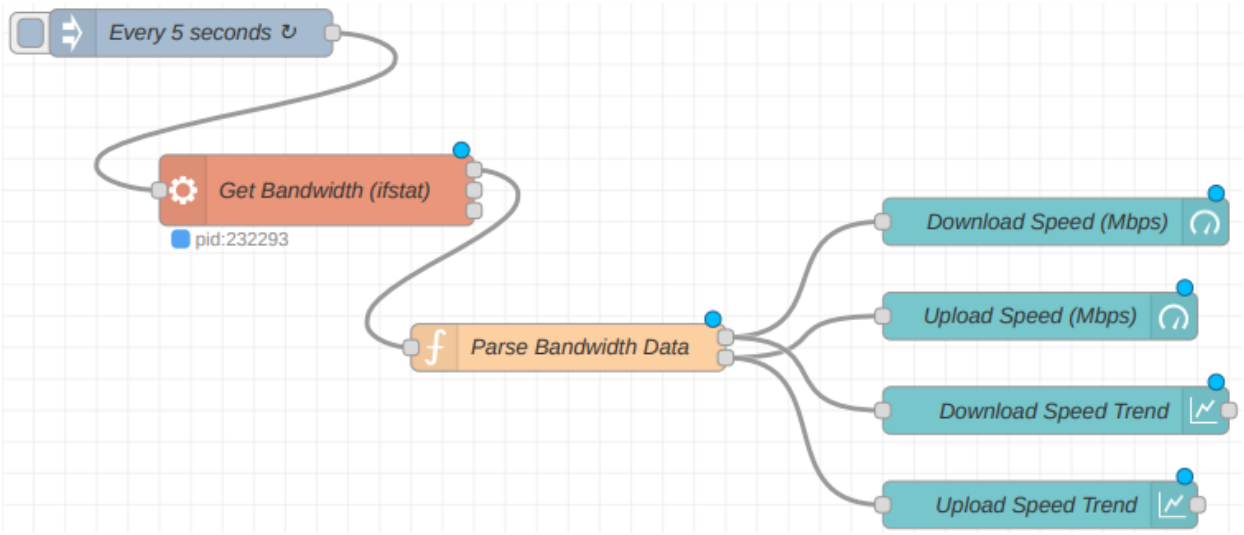


Figure 10 Tab 4 of Bandwidth Usage Nodes Configuration

   a. Activates bandwidth monitoring at regular intervals.

2. Exec Node (Get Bandwidth):
   a. Executes a shell command on the Raspberry Pi to get real-time interface statistics.
3. Function Node (Parsed Bandwidth Data):
   a. Parses and transforms the raw output of the exec node into JSON payloads.
4. UI widgets:
   a. Gauge widgets display the current download and upload speeds.
   b. Chart widgets display bandwidth usage over time, allowing users to identify trends and spikes.

Breaking down its components, ifstat -i eth0 1 1 specifically monitors the Ethernet interface eth0 for a single interval lasting one second, providing instantaneous download and upload rates in kilobytes per second (KBps). The output of this command is then piped to awk 'NR==3{print $1, $2}', which filters for the third line of the ifstat output, where the actual traffic rates are located, and subsequently prints only the first and second fields, representing the Download KBps and Upload KBps, respectively.

SYN Flood Attack Analysis

A SYN flood attack was deliberately launched within the isolated LAN environment to thoroughly evaluate the monitoring system's response to network-level DoS scenarios. This simulation was critical in illustrating how the Node-RED network analytics dashboard responds to substantial network stress and effectively visualises the consequent network degradation in real time. The assault was unique in that it targeted the Raspberry Pi 400, which played a dual duty as both the dedicated monitoring node and the victim of the SYN flood. This intentional design option was made to reduce hardware requirements while demonstrating the system's efficiency and giving a thorough test of its defensive and analytical capabilities against a common network threat. The following Figure 11 presents the results obtained from the SYN Flood Attack.

The attack was launched from the Raspberry Pi using the following command in the terminal:

```
sudo hping3 -S -p 80 192.168.1.102 --flood
```

Table 3 outlines the key command components of the SYN Flood attack, along with their corresponding explanations.

The commands given in Table 3 initiate a high-rate flood of SYN packets, forcing the target device to allocate resources for a rapidly increasing number of half-open TCP connections. Each SYN packet sent to the target's port 80 triggers the allocation of memory and processing power to track the connection state in the TCP backlog queue. As the number of pending connections grows, the target's resources, particularly its connection table and CPU cycles, become saturated.

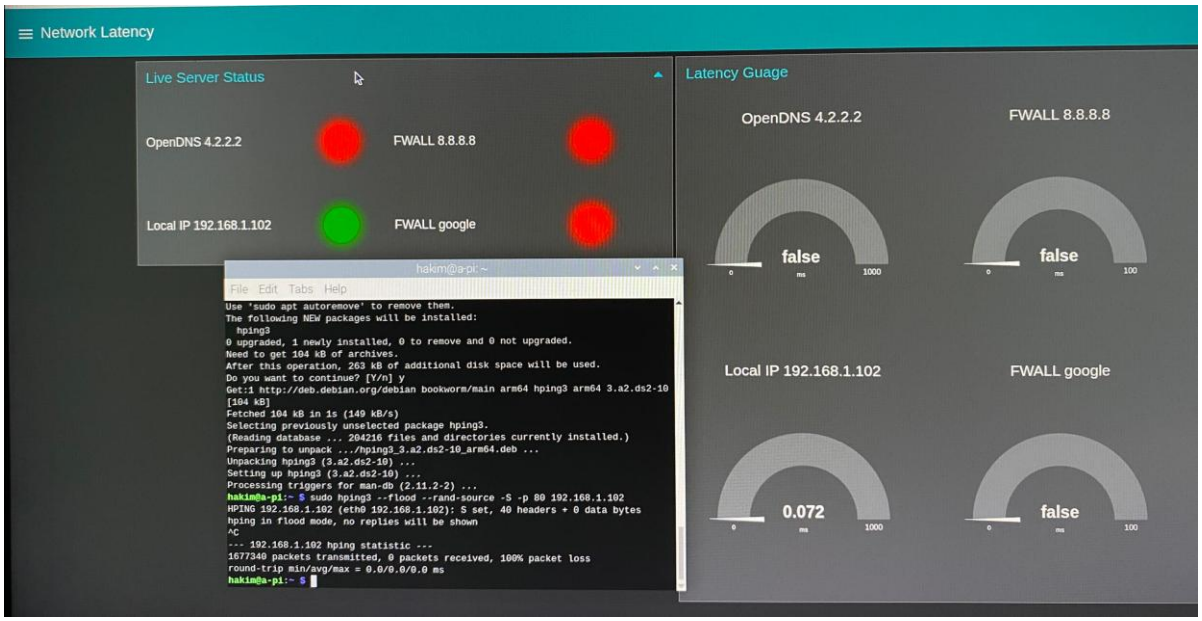


Figure 11 Result of SYN flood attack

Table 3 SYN flood attack command breakdown

Component	Description
`sudo`	Grants root privileges necessary for crafting raw TCP packets.
`hping3`	A packet crafting and testing tool capable of generating SYN flood traffic.
`-S`	Sets the TCP SYN flag, mimicking the initial handshake request of TCP.
`-p 80`	Targets port 80 (HTTP) on the victim device (192.168.1.102).
`192.168.1.102`	Specifies the target IP address (the same Raspberry Pi).
`--flood`	Sends SYN packets as fast as possible without waiting for replies.

Over time, this flood consumes bandwidth and can degrade or completely deny legitimate users' access to the service.

Functional Impact

The SYN flood attack significantly disrupted normal network operations by saturating the outbound traffic queue, which in turn prevented external endpoints from responding to ping requests and overloaded the router's state table with incomplete TCP connections. The Node-RED Network Analytics Dashboard provided clear, multi-dimensional evidence of this disruption. Tab 3, "Latency," immediately showed endpoint unreachability and a series of latency failures, while Tab 4, "Bandwidth," highlighted a correlated and distinct spike in upload throughput.

TCP Port Scan Analysis

A TCP port scan was conducted in an isolated LAN environment to assess the network analytics system's ability to detect reconnaissance activity. Port scanning is a well-documented pre-attack technique that allows adversaries to find active hosts and services with potentially exploitable vulnerabilities. This experiment demonstrates how the dashboard detects network enumeration activities in real-time by utilising the Raspberry Pi 400 as both a scanning agent and a monitoring system.

The command describes the components of an Nmap command used for network scanning. The command utilises sudo to grant administrative rights, which are necessary for raw packet crafting during the scan. Nmap itself is the core network mapping and vulnerability assessment utility. The -sS option executes a TCP SYN scan, also known as a "half-open" scan, for stealthier probing, as it does not complete the full TCP handshake. The -T4 option applies an aggressive timing template, enabling a faster scanning process. Finally, 192.168.1.0/24 specifies the target, instructing Nmap to scan the entire subnet for active host and service discovery. This specific configuration allowed for the scanning of all 256 IP addresses within the LAN and the probing of common TCP ports on any responsive

devices found. The scan successfully identified four active hosts within the subnet, mirroring the results presented in Figure 12. Key findings from the scan output provided crucial insights into the network's composition and potential vulnerabilities.

Specifically, the router (192.168.1.1) was found to expose UPnP on port 1900, which, if improperly configured, can pose significant security risks by allowing devices to automatically open ports to the internet. The Windows Host (192.168.1.101) revealed the presence of SMB-related ports (139, 445) along with ICSLAP, indicating the use of legacy protocols that have historically been targets for malware such as WannaCry. For the Raspberry Pi devices (192.168.1.102 & .103), the scan confirmed the detection of expected services like SSH and HTTPS, validating their intended configurations. Notably, the entire scan completed in a mere 10.15 seconds, demonstrating the speed at which an adversary could map the LAN's architecture and identify exploitable services.

Figure 12 shows the Nmap command input on the Pi CLI. This simulation emphasises the need for early identification of spying efforts. While a port scan may not directly impair network services, it does provide attackers with valuable intelligence, enabling them to methodically identify future vulnerabilities or coordinate lateral movement throughout the network. The dashboard's features, particularly its capacity to detect new active hosts, visualise exposed services, and flag anomalous traffic patterns, are therefore critical. These features work together to ensure that network administrators are immediately notified of such pre-attack actions, enabling quick intervention and mitigation before more severe attacks can be initiated.

Honeypot Attack Simulation

This section describes the simulation of a honeypot scenario in an isolated LAN environment to show how the system can detect misleading services being probed by a possible attacker. Honeypots are intentionally placed systems that entice attackers to interact with them, allowing network defenders to gain intelligence about adversary surveillance and exploitation techniques. A fake vulnerable service was exposed on TCP port 8080 via the Raspberry Pi 400. The

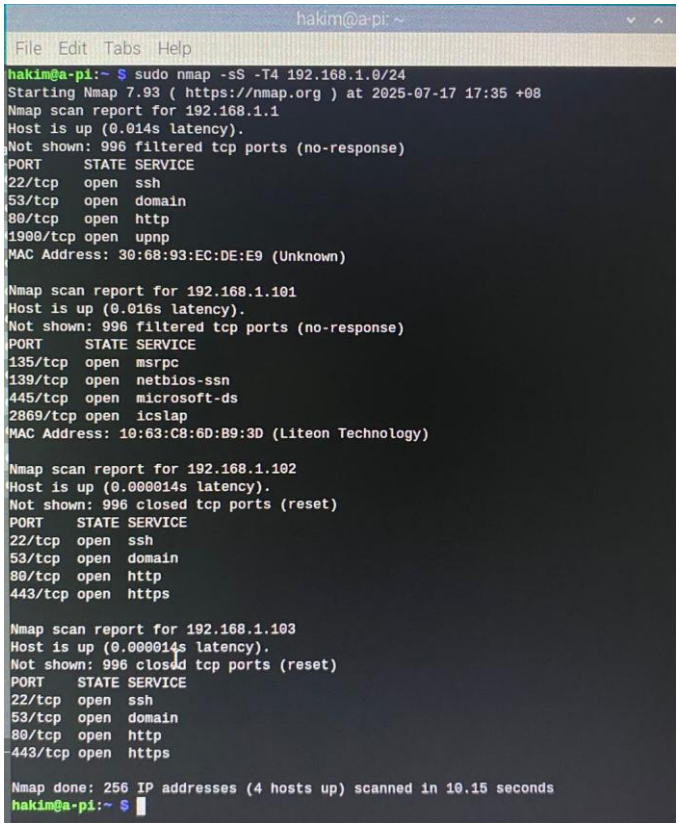


Figure 12 Nmap command on Pi

```
hakim@a-pi:~ $ sudo apt install netcat-openbsd -y
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
netcat-openbsd is already the newest version (1.219-1).
netcat-openbsd set to manually installed.
The following packages were automatically installed and are no longer required
  libbasicusageenvironment1 libgroupsock8 liblivemedia77
  linux-headers-6.6.74+rpt-common-rpi linux-headers-6.6.74+rpt-rpi-2712
  linux-headers-6.6.74+rpt-rpi-v8 linux-image-6.6.74+rpt-rpi-2712
  linux-image-6.6.74+rpt-rpi-v8 linux-kbuild-6.6.74+rpt
Use 'sudo apt autoremove' to remove them.
0 upgraded, 0 newly installed, 0 to remove and 3 not upgraded.
hakim@a-pi:~ $ sudo nc -lvkp 8080
Listening on 0.0.0.0 8080
Connection received on 192.168.1.102 46968
Connection received on 192.168.1.103 53714
^C
hakim@a-pi:~ $ sudo nmap -sV -p 8080 192.168.1.102
Starting Nmap 7.93 ( https://nmap.org ) at 2025-07-17 20:44 +08
Nmap scan report for 192.168.1.102
Host is up (0.00011s latency).

PORT     STATE  SERVICE    VERSION
8080/tcp closed http-proxy

Service detection performed. Please report any incorrect results at https://nma
.org/submit/ .
Nmap done: 1 IP address (1 host up) scanned in 1.26 seconds
hakim@a-pi:~ $
```

Figure 13 Honeypot attacker scenario

Node-RED network analytics dashboard (Tab 1) was then utilised to track and assess attacker behaviour in real time.

Figure 13 shows the honeypot attacker scenario. The Raspberry Pi was configured to listen for TCP connections on port 8080 using netcat-openbsd. This created a fake HTTP service, making the device appear as if it were hosting a web application. The attacker, simulated from the same Raspberry Pi, performed an Nmap scan targeting the honeypot port. The command used incorporates several key options to achieve its scanning objective. The -sV option enables service detection mode, which is crucial for identifying application banners and versions running on the target port. The -p 8080 option specifically focuses the scan on port 8080, which is the honeypot's exposed port. Finally, 192.168.1.102 specifies the IP address of the honeypot system as the target for the scan.

Tab 1: Nmap Scan Visualisation after honeypot attack

Figure 14 illustrates Tab 1 visualisation of Nmap scan results after the honeypot attack, providing insights

nmap Scan Visualization

IP Address	Host Status	Port	Protocol	Port State
192.168.1.1	up	22	tcp	open
192.168.1.1	up	53	tcp	open
192.168.1.1	up	80	tcp	open
192.168.1.1	up	1900	tcp	open
192.168.1.100	up	62078	tcp	open
192.168.1.102	up	22	tcp	open
192.168.1.102	up	53	tcp	open
192.168.1.102	up	80	tcp	open
192.168.1.102	up	443	tcp	open
192.168.1.102	up	8080	tcp	open
192.168.1.103	up	22	tcp	open
192.168.1.103	up	53	tcp	open
192.168.1.103	up	80	tcp	open
192.168.1.103	up	443	tcp	open
192.168.1.103	up	8080	tcp	open

Figure 14 Nmap tab after honeypot attack

into detected activity and its representation on the dashboard.

Figure 14 depicts how the Nmap Scan Visualisation page effectively revealed the honeypot's port as open during the active scan. Specifically, Host 192.168.1.102 was shown, with TCP port 8080 readily open. Subsequent scans indicated that port 8080 was also open on Host 192.168.1.103, indicating that the network contained numerous honeypot endpoints. These entries displayed dynamically on the dashboard when the attacker engaged with the fraudulent service, providing obvious evidence of reconnaissance efforts focused on a high-numbered port.

Functional Analysis of Honeypot Attack

This simulation effectively illustrates the seamless integration of a lightweight honeypot, even one without advanced logging capabilities, with the Raspberry Pi's monitoring system to provide crucial indicators of reconnaissance. The detection of unexpected open ports, such as port 8080, allows administrators to promptly verify whether the exposure is an intentional setup (like a honeypot) or an unintentional configuration oversight.

Furthermore, by utilising a simple Netcat listener, this test demonstrates how quickly and inexpensively a basic honeypot can be deployed for effective threat hunting within small LAN environments. This scenario aligns strongly with Zero Trust principles; honeypots function as strategic traps within the network, enabling defenders to actively identify unauthorised scanning and probing attempts that violate established trust boundaries. By continuously validating which ports are exposed and diligently monitoring interactions with these deceptive services, the system enforces strict network visibility and significantly aids in isolating potentially compromised devices.

Calability and Resource Management

The proposed system was designed for small to medium-sized LAN environments; however, its architecture supports scalability with appropriate adaptations. Node-RED's modular, event-driven design allows monitoring components to be replicated or extended without restructuring the core system. For larger networks, multiple Raspberry Pi devices can be deployed as distributed monitoring nodes, each responsible for a specific subnet, with aggregated metrics forwarded to a central dashboard using lightweight messaging protocols. Higher traffic loads can be managed by adjusting scan frequencies, rate-limiting active probes such as Nmap and hping3, and prioritising passive monitoring to minimise overhead. While the current implementation targets resource-constrained environments, these design characteristics provide a clear pathway for scaling the system to larger LAN deployments.

CONCLUSIONS

This study successfully designed and implemented a lightweight, Raspberry Pi-based network threat analytics system capable of real-time monitoring and detection of anomalous activities within a LAN. By leveraging open-source tools such as Node-RED, Nmap, hping3, and Netcat, the system demonstrated its ability to visualise and analyse network behaviours across multiple dimensions. The deployment of a multi-tab Node-RED dashboard, comprising Nmap Scan Visualisation, Network Health, Network Latency, and Bandwidth Usage, enabled holistic monitoring of the LAN environment, with each tab addressing distinct aspects of network security. The system was validated through the simulation of three attack scenarios: a SYN Flood Attack, which illustrated the detection of high-volume DoS traffic; a TCP Port Scan, which showcased early recognition of adversarial reconnaissance efforts; and a Honeypot Attack, which demonstrated the system's ability to identify probing of deliberately exposed fake services. Future enhancements could include integrating machine learning algorithms for predictive analytics and anomaly detection, enabling the system to anticipate threats before they escalate.

ACKNOWLEDGEMENT

The authors would like to acknowledge Universiti Teknologi PETRONAS and the Department of Computing for their support of this work.

REFERENCES

[1] M. N. Osman, K. A. Sedek, N. A. Othman, M. A. Rosli, and M. Maghribi, "Enhancing security and privacy in local area network (LAN) with TORVPN using Raspberry Pi as access point: a design and implementation," *Journal*

of Computing Research and Innovation, vol. 6, no. 2, pp. 29–40, 2021.

[2] L. Diana, P. Dini, and D. Paolini, "Overview on Intrusion Detection Systems for Computers Networking Security," Computers, vol. 14, p. 87, 2025, doi: 10.3390/computers14030087.

[3] R. Wainwright, M. Bagheri, A. Salama, and R. Saatchi, "Software-defined networking security detection strategies and their limitations with a focus on distributed denial-of-service for small to medium-sized enterprises," *Appl. Sci.*, vol. 15, no. 23, p. 12389, 2025, doi: 10.3390/app152312389.

[4] M.M. Aslam, A. Tufail, R.A.A.H. Mohd Apong, L.C.D. Silva, and M.T. Raza, "Scrutinising Security in Industrial Control Systems: An Architectural Vulnerabilities and Communication Network Perspective," IEEE Access, vol. 12, pp. 67537–67573, 2024.

[5] Y. Zhang, B. Asulba, N. Schumacher, M. Sousa, P. Souto, L. Almeida, P. Santos, N. Martins, and J. Sousa, "Implementing and deploying an ml pipeline for iot intrusion detection with node-red," in Proc. Cyber-Physical Systems and Internet of Things Week 2023, 2023, pp. 247–253.

[6] S. Tripathi and R. Kumar, "Raspberry Pi as an Intrusion Detection System, a Honeypot and a Packet Analyzer," in 2018 Int. Conf. Computational Techniques, Electronics and Mechanical Systems (CTEMS), 2018.

[7] N. Ahmad and M. Habib, "Analysis of network security threats and vulnerabilities by development and implementation of a security network monitoring solution," M.S. thesis, Blekinge Institute of Technology, 2010. [Online]. Available: http://www.bth.se/fou/cuppsats.nsf

[8] Node-RED Project, "A set of dashboard nodes for Node-RED," ver. 3.6.6, OpenJS Foundation, 2024. [Software]. [Online]. Available: https://flows.nodered.org/node/node-red-dashboard

[9] I.U. Onwuegbuzie, "Node-RED and IoT analytics: A real-time data processing and visualization platform," *Tech Sphere Multidisciplinary Int. J. (TSMIJ)*, vol. 1, no. 1, pp. 1–12, 2024, doi: 10.5281/zenodo.13856860.

[10] H. Altaf et al., "Enhancing LAN security by mitigating credential threats via HTTP packet analysis with Wireshark," *J. Comput. Biomed. Informatics*, vol. 6, no. 2, pp. 433–440, 2024.

[11] G.B.M.T. Sitorus, R.A.G. Efendi, J.F. Napitupulu, H. Pranoto, and E.S. Hermawan, "Bridging the Gap: A Comparative Evaluation of Agile, Waterfall, and Hybrid Methodologies in Modern Software Projects," *Procedia Computer Science*, vol. 269, pp. 1259–1268, 2025.