An Empirical Analysis of MeshShield: a Network Security System for Fully Distributed Networks

Selina Shrestha[1], Willian T. Lunardi[1], and Martin Andreoni[1]

[1]Technology Innovation Institute, 9639 Masdar City, Abu Dhabi, UAE
{selina.shrestha, willian.lunardi, martin.andreoni}@tii.ae

Abstract

Distributed networks, such as wireless mesh networks, offer scalability, fault tolerance, and cost-effectiveness but face security challenges due to their decentralized nature. Traditional security approaches are inadequate, necessitating the adoption of the Zero Trust security model. This paper introduces MeshShield, a security system for wireless mesh networks. It implements mutual authentication, continuous authentication, a decision engine, and node quarantine in a decentralized manner to enhance network resilience and mitigate risks. It ensures that only authenticated and authorized nodes join the network and enables periodic verification of device behavior. Security information is shared among nodes, enabling a global view of network security. Malicious behavior detection triggers rapid response measures, including broadcasting announcements and quarantining malicious nodes. Our evaluation demonstrates that MeshShield has minimal impacts on network performance (3.46% drop in throughput) and fairly low average CPU, and memory usage of 14.96%, and 1.38% respectively, making it suitable for resource-constrained devices. Compared to existing approaches, MeshShield integrates security features in a distributed manner, providing an integrated system to safeguard sensitive data in wireless mesh networks with nominal effects on network performance.

1 Introduction

Distributed networks have gained significant attention due to their inherent scalability, fault tolerance, and cost-effectiveness. Wireless mesh network is an applied example of such a network. Mesh networks consist of interconnected nodes that communicate with each other to establish network connectivity without relying on a centralized infrastructure. While these networks provide significant advantages, such as easy deployment and dynamic topology, they introduce unique security challenges due to their decentralized nature and wireless communication medium [1]. Traditional security approaches, such as perimeter-based defenses, are ineffective in this dynamic and decentralized environment. Consequently, the Zero Trust security model has emerged as a promising paradigm to address these vulnerabilities and protect sensitive information.

The Zero Trust principle can be effectively applied to secure wireless mesh networks by treating every node as untrusted, requiring continuous verification and validation before granting access to resources. This approach ensures that each node in the mesh network is subject to authentication and authorization checks, limiting the potential for unauthorized access and lateral movement within the network. By implementing Zero Trust principles, wireless mesh networks can enhance their resilience and mitigate the risks associated with their decentralized and wireless nature.

This paper introduces MeshShield, which integrates security sensors to protect mesh networks. With mutual authentication, the MeshShield ensures that only authorized nodes can join the mesh network, reducing the risk of unauthorized devices compromising network integrity. Secondly, following the Zero Trust paradigm, continuous authentication allows a periodic verification of the behavior of the devices. All the nodes contain a security table with the current security information about their neighbors, which is exchanged periodically to have a global view of the security in the network. A decision engine processes all the information from continuous authentications, adding a trust degree for each node. Then, when malicious behavior is detected, announcements are broadcasted, enabling rapid response to protect the wireless mesh network. Finally, each node can set the malicious node in quarantine, blocking its network traffic. These benefits collectively enhance the security posture of wireless mesh networks, safeguarding sensitive data and minimizing the potential impact of attacks. We evaluated the proposed security system and showed that even on resource-constrained devices with limited processing power and memory, MeshShield has a low impact on the network performance and the resource consumption of the device.

The remainder of the paper is organized as follows. In Section 2, we describe the system architecture for MeshShield. The proposal is evaluated, and the results are discussed in Section 3. Section 4 lists related works. Finally, Section 5 concludes the article.

2 Proposed System - MeshShield

Our proposed system addresses various threats that can compromise the security of wireless mesh networks. The system detects and mitigates three primary threat models: intrusion, fake/rogue nodes, and impersonation. In the case of intrusion, MeshShield identifies nodes attempting to access the network without valid credentials. These unauthorized nodes are swiftly detected and prevented from compromising network integrity. Additionally, the system detects fake or rogue nodes that possess valid credentials but lack valid certificates. By closely examining the certificates associated with each node, MeshShield effectively identifies and neutralizes these deceptive entities. Furthermore, the system can identify and counteract impersonation attacks, where a node attempts to masquerade as another legitimate node within the network. Through robust authentication mechanisms and continuous monitoring, MeshShield ensures the accurate identification of node identities, preventing malicious actors from successfully impersonating other network entities. With its comprehensive threat detection capabilities, MeshShield significantly enhances the security posture of wireless mesh networks by effectively countering intrusion, fake/rogue nodes, and impersonation threats.

In the proposed system, the root certificate is assumed to be securely pre-distributed at all nodes during provisioning. Additionally, a pair of prime field Weierstrass 256-bit Elliptic Curves (also known as secp256r1P-256) keys is generated at each node. The Elliptic Curve (EC) key pairs are generated and stored on softHSM using the pkcs11-tool. The public key is exported from the Hardware Security Module (HSM) to a DER certificate file. Different components of the MeshShield system are described below.

2.1 Distributed Mutual Authentication

This component ensures that only nodes with valid credentials, i.e., a valid root certificate, can access the mesh network. Since no central authority exists, a distributed mutual authentication is performed when a node wants to join the network. During this, nodes already in the mesh act as servers to authenticate the newly joining nodes, which act as clients. Each server has a wireless access point named $AuthAP\_ID$, where ID is the node's mesh ID. A client scans for available access points with the pattern "AuthAP" and connects to the one that provides the best signal quality. The mutual authentication then takes place between the selected server and the client. It should be noted that each node uses two different radios: one acting as an access point to facilitate mutual authentication of new nodes and the other for mesh connectivity. Details of the steps carried out during the authentication are given below.

Step 1: Authentication Message Generation (Client Side). The client generates the authentication message using Algorithm 1, where *rootcert* refers to the root certificate and *myPrivKey* and *myID* refer to the node's own (i.e., the client's) private key and ID respectively. The message consists of the hash digest of the client's local root certificate signed by the client's private EC key using the Elliptic Curve Digital Signature Algorithm (ECDSA), the client's public EC key, the client's ID, a nonce, and timestamp of the computed message. A nonce is a random number used only once and, in combination with the timestamp ts, is used to prevent a replay attack. Also, a Hash-based Message Authentication Code (HMAC) of the message, computed using the root certificate as key, is appended to ensure the message integrity at the receiver.

Algorithm 1 Authentication Message Generation

1: Compute the hash digest of the local root certificate:
   $digest_{rootcert}$ = Hash($rootcert$)
2: Sign the hash digest of the root certificate using ECDSA:
   $sign_{rootcert}$ = ECDSA$_{sign}$($myPrivKey, digest_{rootcert}$)
3: Compute $hmac$ = HMAC$_{rootcert}$($sign_{rootcert}, myPubKey, myID, nonce, ts$)
4: $message$ = $\{sign_{rootcert}, myPubKey, myID, hmac, nonce, ts\}$
5: Send $message$

Step 2: Verify received authentication message (Server Side). Once the server receives the authentication message from the client, the message is verified according to Algorithm 2, where *nodepubKey* and *nodeID* refer to the other node's (i.e., client's) public key and ID respectively. First, a fresh HMAC is computed using the server's root certificate as a key to verify the message's integrity and that the message was generated by a valid client (with the same root certificate). Also, the nonce and timestamp of the message are verified to retrieve a fresh message. Then, the received ECDSA signature of the root certificate is verified using the received client public key and a freshly computed hash digest of the server's root certificate. If the verification passes, the client is deemed authentic, and the next steps for mesh connection are carried out. Otherwise, the client is not valid, and the connection is closed.

Algorithm 2 Authentication Message Verification

1: Received message:
   $message$ = $\{sign_{rootcert}, nodePubKey, nodeID, hmac, nonce, ts\}$
2: Compute fresh hmac using local root certificate as:
   $hmac'$ = HMAC$_{rootcert}$($sign_{rootcert}, nodePubKey, nodeID, nonce, ts$)
3: **if** $hmac'$ = $hmac$ **then**
4: Received message was generated by a valid node, and it was not altered during transmission
5: Continue to the next steps
6: **else**
7: Invalid hmac
8: Authentication failed, close connection to the node
9: **end if**
10: Compute the hash digest of the local root certificate:
   $digest'_{rootcert}$ = Hash($rootcert$)
11: Verify received ECDSA signature of root certificate using the node's received public key and freshly computed hash digest of the local root certificate:
   $verification$ = ECDSA$_{verify}$($nodePubKey, digest'_{rootcert}, sign_{rootcert}$)
12: **if** $verification$ = true **then**
13: Received signature and root certificate are valid
14: Continue to the next steps
15: **else**
16: Invalid signature and root certificate
17: Authentication failed, close connection to the node
18: **end if**

Step 3: Authentication Message Generation (Server Side). Steps mentioned in Algorithm 1 are repeated at the

server, and the message generated (which includes the signed root certificate, server's public key, and ID) is sent to the client for authentication at the client's side.

Step 4: Verify received authentication message (Client Side). Steps mentioned in Algorithm 2 are repeated at the client to validate the server's signature and root certificate to authenticate the server.

Step 5: Send encrypted mesh password to the client (Server Side). The mesh password or key required for a node to join the network is securely shared with the authenticated client according to Algorithm 3. The password can have been pre-distributed before; otherwise, a random number is generated on the server side. This password, *password*, is encrypted with a symmetric shared secret key, *secret*, derived from the server's private and client's public keys using the Elliptic Curve Diffie-Hellman (ECDH) algorithm. The encrypted password, *encrypted_password*, is sent to the client.

Algorithm 3 Mesh password encryption

1: Derive a symmetric shared key from the client and server EC keys:
   $secret = \text{ECDH}_{derive}(ServerPrivKey, ClientPubKey)$
2: Encrypt the mesh password using the derived symmetric key:
   $encrpted\_password = \text{SHA256Encrypt}(secret, password)$
3: Send *encrpted_password* to client

Step 6: Decrypt the mesh password and connect to the mesh (Client Side). At the client, the symmetric shared secret key is derived from the client's private and the server's public keys. Then the received password is decrypted, and the client connects to the mesh network as described in Algorithm 4.

Algorithm 4 Mesh password decryption

1: Derive a symmetric shared key from the client and server EC keys:
   $secret = \text{ECDH}_{derive}(ClientPrivKey, ServerPubKey)$
2: Decrypt the mesh password using the derived symmetric key:
   $password = \text{SHA256Decrypt}(secret, encrypted\_password)$
3: Connect to the mesh network using *password* as key

Step 7: Exchange mesh's IP and MAC addresses and add the node to Security Table (Both Side). The client and server exchange their respective mesh IP and MAC addresses. The server stores the client's ID, mesh IP, and mesh MAC in its security table while the client does the same with the server's credentials, indicating that the nodes were mutually authenticated.

Step 8: Start access point to facilitate mutual authentication of new incomer (Client Side). Now, the client starts a wireless access point named *AuthAP_myID* to facilitate mutual authentication of new nodes. Thus, distributed mutual authentication ensures that the mesh password is securely shared only with authorized nodes possessing a valid root certificate, denying any unauthorized node from intruding into the network.

2.2 Security Beat

Once mutual authentication is completed and the node joins the mesh, it waits for a security beat. The security beat (SecBeat) regularly monitors the system's security and is broadcasted every *sec_beat_time*. A newly joined node waits for a security beat for one *sec_beat_time*. If a security beat is received, it starts the security beat execution. If no security beat is received for a whole *sec_beat_time* period, it will broadcast a security beat message through the mesh network and start the security beat.

The execution of a security beat consists of five main components, as shown in Figure 1, which work together to ensure the security of the MeshShield system beyond the initial authentication. First, three rounds of continuous authentications according to a lightweight mechanism are carried out with one-hop neighbors of a node. Then, the local security tables updated with the continuous authentication results at each node are exchanged with other nodes in the mesh so that all nodes have a global view of the network's security status. This is then analyzed by the decision engine at each node to determine any malicious behavior. If any node is found to be malicious, the deciding node announces this to all the nodes in the network via the Malicious Behavior Announcement (MBA) broadcast message. Then, the malicious node is put into quarantine, where it is blocked from accessing the mesh network.

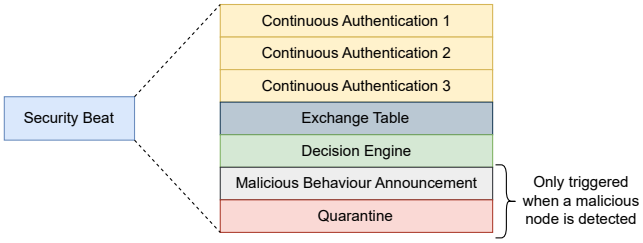


Figure 1: Components of Security Beat

2.2.1 *Continuous Authentication (CA)*

Through this component, each node in the mesh verifies the identity of its one-hop neighbors to check if those are still the initially authorized nodes during mutual authentication. This is accomplished using a lightweight Continuous Authentication (CA) protocol proposed in [7] that uses time-bound tokens generated using a shared secret, a time-varying component, and a random value for each authentication round. This mechanism uses low-complexity hash and Message Authentication Code (MAC) operations, making it suitable for resource-constrained mesh devices. During verification, the time-varying tokens are valid only for a specific time frame and linked to the shared secret at the server node. The shared secret between any pair of nodes is derived from their EC keys (exchanged during mutual authentication) using the ECDH mechanism. During the first security beat, neighboring nodes that did not mutually authenticate with each other exchange their public keys, IDs, mesh IPs, and MAC addresses. Henceforward, a unique shared secret key derived from the initially exchanged key pairs is used for each pair of nodes to generate the CA token. This ensures that any impersonation attack from a node with access to the mesh and a fake/rogue node with invalid keys is swiftly detected.

Each node (acting as a server) requests to authenticate its one-hop neighbors (acting as clients) for every round of CA. The nodes respond to requests to authenticate by generating an authentication message according to Algorithm 5.

Algorithm 5 Continuous Authentication Message Generation (Client Side)

1: $secret$ = ECDH$_{derive}(ClientPrivKey, ServerPubKey)$
2: Select random number x_i
3: Compute share, $u_i = secret + timeflag_i + x_i$
4: Compute share authenticator, sa_i = Hash(x_i)
5: Compute mac_i = HMAC$_{secret}(ClientID, ServerID, u_i, timeflag_i)$
6: $message_i = \{ClientID, ServerID, u_i, timeflag_i, sa_i, mac_i\}$
7: Compute crc_i, cyclic redundancy check bits for $message_i$
8: Send $message_i$ appended with crc_i

The server node then performs a series of verification on the received authentication message according to Algorithm 6 to determine the authenticity of the client nodes.

Algorithm 6 Continuous Authentication Message Verification (Server Side)

1: $secret$ = ECDH$_{derive}(ServerPrivKey, ClientPubKey)$
2: Cyclic Redundancy Check (CRC) to check data integrity
3: Check message freshness
4: Check that the received share is fresh (not used previously during the session)
5: Compute fresh MAC as:
   $mac'_i = HMAC_{secret}(ClientID, ServerID, u_i, timeflag_i)$
   and verify that $mac'_i = mac_i$
6: Compute fresh share authenticator as:
   $sa'_i = Hash(u_i - secret - timeflag_i)$
   and verify that $sa'_i = sa_i$
7: **if** All the checks pass **then**
8: Authentication result = pass
9: **else**
10: Authentication result = fail
11: **end if**

During each security beat, three rounds of CA are performed. For any neighbor node, if most of the three CA results are "pass", the final CA result, *CA_Result*, during that security beat is "pass". Else, it is "fail". *CA_Result* and *CA_Server*, the verifying node's ID (i.e., the node's ID), are entered in the security table. Table 1 shows an example of the security table after CA at node N_1, given that N_2, N_3, and N_4 are its one-hop neighbors.

Table 1: Example of security table after CA at node N_1 with neighbors N_2, N_3, and N_4

ID	MAC	IP	CA_Result	CA_Server
N_2	...	10.10.10.2	pass	N_1
N_3	...	10.10.10.3	pass	N_1
N_4	...	10.10.10.4	pass	N_1

2.2.2 Exchange Table

This component exchanges the security tables at each node, containing the continuous authentication results of the node's neighborhood, with nodes throughout the mesh network using Algorithm 7, so that all nodes will have a global security table at the end. The algorithm adds columns described in Table 2 to the security table to compute the exchange table. The exchange table is initially computed by entering the node's IP in the "Source_ID" column and the IPs of the node's one-hop neighbors in the "To_Send" column. Then, the IPs in "To_Send" are appended to "Destination_IP," and the table entries are sent to the IPs in "To_Send". Initially, a server socket is created, which puts any table entry received from another node into a received table queue. Now, items in the queue are popped and concatenated to the node's current exchange table. This completes one round of the table exchange. For the next exchange, "To_Send" is again computed for each table entry as the node's one-hop neighbors, not in "Source_IP" or "Destination_IP". This enables the algorithm to keep track of nodes that have received a particular table entry so that a node only forwards it to neighboring nodes that have not already received it. In doing so, the algorithm avoids wastage of bandwidth caused by redundant exchanges in multi-hop scenarios. Then, the table entries with non-empty "To_Send" columns are again sent to the respective nodes, and any items in the received queue are popped and concatenated to the current table. This is repeated until no data is received from other nodes and no IP is in the "To_Send" column. At this point, all nodes will have a global security table that contains local continuous authentication results generated at every node in the network. The maximum rounds of exchanges the algorithm requires is equal to the number of hops in the mesh network.

Table 2: Additional columns in Exchange Table

Column	Description
Source_IP	IP of the node that generated the table entry, i.e., the node's own IP when it generates the exchange table initially
Destination_IP	IPs of nodes where the table entry has been sent
To_Send	IPs of neighboring nodes where the table entry is to be sent next computed as mesh neighbors, not in (Source_IP, Destination_IP)

Table 3 shows an example of a global security table constructed by the end of table exchange for a mesh topology with nodes N_1, N_2, N_3, N_4, where N_1, N_2, and N_3 are in the same neighborhood, and N_4 is one hop away from N_1 but two hops away from N_2 and N_3.

Table 3: Example of global security table

ID	MAC	IP	CA_Result	CA_Server
N_2	...	10.10.10.2	pass	N_1
N_3	...	10.10.10.3	pass	N_1
N_4	...	10.10.10.4	pass	N_1
N_1	...	10.10.10.1	pass	N_2
N_3	...	10.10.10.3	pass	N_2
N_1	...	10.10.10.1	pass	N_3
N_2	...	10.10.10.2	pass	N_3
N_1	...	10.10.10.1	pass	N_4

2.2.3 Decision Engine

The decision engine runs at each node and analyzes the global security table according to Algorithm 8 to determine

Algorithm 7 Exchange Security Table

```
Create a server socket that puts received messages into a "received table queue"
Compute initial exchange table by adding columns: "Source_IP" = node's self IP, "Destination_IP" = "", and "To_Send" = mesh neighbor IPs, to the node's security table
while data is received from other nodes (i.e., server socket does not timeout) or there are nodes in "To_Send" do
    Send table entries to neighbors in "To_Send":
        Append IPs in "To_Send" to "Destination_IP"
        Send respective table entries to IPs in "To_Send" column
        Clear "To_Send"
    if "received table queue" is not empty then
        Pop the contents of the queue and concatenate them to the exchange table
        Compute IPs to send for the next exchange as: mesh neighbors not in ("Source_IP", "Destination_IP")
    end if
end while
Drop columns "Source_IP", "Destination_IP" and "To_Send" and use the resulting global table as input to the decision engine
```

if other nodes in the network are benign or malicious. The global table can be perceived as a collection of votes where every node, as a CA Server, votes for the authenticity of its one-hop neighbors through CA results. Thus, for each unique node ID in the table, the decision engine checks what its various CA Servers have reported. If most CA Results are "fail", the node is marked as malicious. Else, the node is marked as benign. For nodes determined as malicious by the decision engine, malicious behavior announcements and quarantine are triggered.

Algorithm 8 Decision Engine

```
for each unique node ID in the global table do
    if more than half of the CA_Servers have reported CA_Result as "fail" then
        Node is malicious
    else
        Node is benign
    end if
end for
```

2.2.4 Malicious Behavior Announcement

If any node's decision engine determines a mesh node to be malicious, malicious behavior announcements with the node's ID, MAC and IP are broadcasted throughout the mesh network. This notifies nodes across the mesh of a node's malicious behavior.

2.2.5 Quarantine

Then, a malicious node is put into quarantine for a predetermined period. During this time, all network traffic to or from the malicious node is blocked, disabling it from communicating over the mesh. After the quarantine, the node is unblocked, and its authenticity is rechecked during the following security beat.

3 Results

Our proposal is evaluated on Raspberry-pi 4B boards, each acting as a node in the mesh network. Each board uses a Broadcom BCM2711 (ARM v8) 64-bit SoC 1.5GHz and 8GB of RAM. The boards run Ubuntu 20.04 and execute the B.A.T.M.A.N IV mesh routing protocol on the top. We use SparkLan WUBR-508N USB dongle with an rt2800 driver supporting 802.11s mode to establish the mesh network.

3.1 Network Performance

The throughput achieved, given by `iperf` tests, while the security beat of MeshShield is running (SecBeat mode) vs. when the protocol is not running (Neutral mode) is compared to assess the bandwidth consumed by the security beat. The *sec_beat_time* is set to 2 minutes. The experiments are evaluated for User Datagram Protocol (UDP) transmissions throughout three security beats (i.e., 6 minutes).

In the first experiment, we evaluate the scalability of MeshShield, analyzing the Throughput Vs. The number of Neighboring Nodes. The throughput is measured for a varying number of nodes in a neighborhood (i.e., all nodes are within one hop distance from each other) as continuous authentications during SecBeat are performed between one-hop neighbors. Figure 2a shows the throughput achieved in Neutral and SecBeat modes over two to six nodes. It is observed that the throughput in both modes remains fairly constant as the number of nodes increases, with SecBeat resulting in an average of 7 Mbps (3.46%) less throughput. The uniform difference between the throughput achieved in Neutral vs. Secbeat mode demonstrates that the protocol is scalable regarding bandwidth consumption.

Next, we analyze the Throughput Vs. Number of Hops. A mesh topology consisting of five nodes, $N_1, N_2, \ldots N_5$, where each node N_n is one hop away from its adjacent nodes N_{n-1} and N_{n+1}, is considered. Hence, the network consists of four hops where N_1 is one hop away from N_2, two hops away from N_3, and similarly, four hops away from N_5. Figure 2b presents the throughput of Neutral and Secbeat modes across varying hops, achieved by taking measurements between N_1 and the rest of the nodes consecutively. The throughput for Neutral and SecBeat modes falls as the number of hops between the nodes increases. While this behavior is expected, it should be noted that for more than two hops, SecBeat's performance is comparable to the Neutral mode.

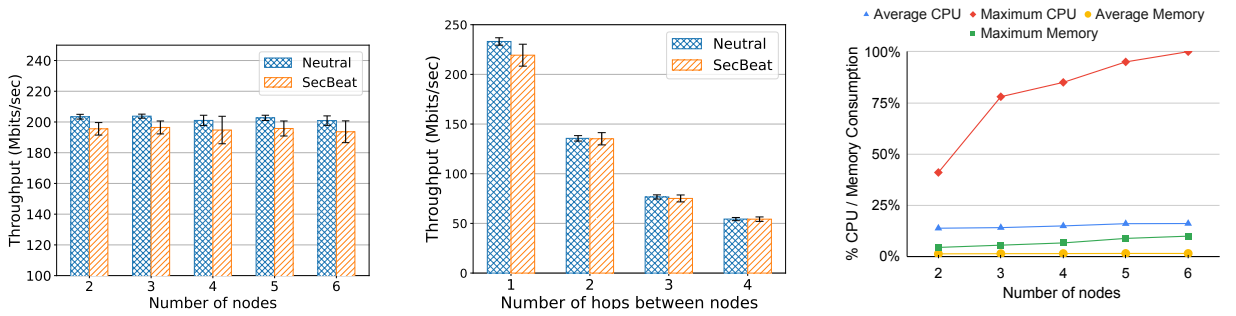
3.2 CPU and Memory Consumption

Table 4 presents the average and maximum % CPU and % memory consumed at the server and client sides during mutual authentication. It is observed that the average CPU and memory consumption at the client and server sides are fairly similar and low in values. While the memory consumption caps at 2.20%, the CPU consumption reaches up to 27%.

Table 4: CPU and Memory Consumption during Mutual Authentication

	Server	**Client**
Average % CPU consumption	3.39%	3.80%
Maximum % CPU consumption	26.00%	27.00%
Average % Memory consumption	1.22%	1.40%
Maximum % Memory consumption	2.20%	2.20%

Figure 2c shows the percentage of CPU and memory consumed at a node during security beat for varying number of nodes in a neighborhood. The measurements are taken



(a) Throughput for Neutral vs. Secbeat mode for a varying number of nodes in a neighborhood

(b) Throughput for Neutral vs. Secbeat mode across a varying number of hops

(c) CPU and Memory consumption during Security Beat for a varying number of nodes in a neighborhood

Figure 2: Performance evaluation of the MeshShield system. a) Network throughput while increasing the number of neighbor nodes, b) Network throughput while increasing the number of hops, c) Resource consumption analysis.

throughout three security beats, where a *sec_beat_time* of two minutes is considered. It is observed that the average CPU and memory consumption remain fairly constant, increasing very slightly as the number of nodes increases from two to six. The SecBeat consumes an average of about 14.96% CPU and 1.38% memory. The maximum memory consumed is fairly low and increases with a uniform slope of 1.1% per additional node. However, the maximum instantaneous CPU consumed is relatively high and increases as the number of nodes increases, owing to the increase in the number of continuous authentications and subsequent computations. Nevertheless, the constant average CPU for varying number of nodes indicates that the spike is short-lived.

4 Related Works

Many security mechanisms have been proposed to protect Wireless Mesh Networks (WMN) [6]. These mechanisms can be classified into two main categories. First, security mechanisms at the network layer are designed to protect the network infrastructure from unauthorized access and malicious attacks. This category includes authentication [3], authorization [5] access control and trust model [2]. Then, security mechanisms at the application layer are designed to protect the data transmitted over the WMN. These mechanisms include encryption [4], data integrity verification, and intrusion detection [8]. Compared with the previous works, the main advantage of our proposal is the combination, integration, and real implementation of all the security features working together in a distributed manner to protect the network infrastructure, making security the main aspect of execution, with minimal impact on the network performance.

5 Conclusions

We presented MeshShield, a comprehensive security system designed to protect distributed networks, specifically wireless mesh networks. MeshShield incorporates several security sensors, including mutual authentication, continuous authentication, and a decision engine, to enhance network resilience and mitigate risks. Sharing security information among nodes allows for a global view of network security, facilitating the detection of malicious behavior. MeshShield swiftly activates measures such as broadcasting announcements and quarantining malicious nodes. Through real experiments, we also demonstrate that MeshShield has minimal impact on network performance and fairly low utilization of resources such as CPU and memory. This makes MeshShield well-suited for resource-constrained mesh devices. In future work, we plan to add new security sensors, such as a network intrusion detection system, and obtain data from Physical layer to enhance the entire system's security.

6 References

[1] M. Andreoni Lopez, M. Baddeley, W. T. Lunardi, A. Pandey, and J. Giacalone. Towards secure wireless mesh networks for uav swarm connectivity: Current threats, research, and opportunities. In *2021 17th International Conference on Distributed Computing in Sensor Systems (DCOSS)*, pages 319–326, Los Alamitos, CA, USA, jul 2021. IEEE Computer Society.

[2] L. H. G. Ferraz, P. B. Velloso, and O. C. M. Duarte. An accurate and precise malicious node exclusion mechanism for ad hoc networks. *Ad hoc networks*, 19:142–155, 2014.

[3] Y. Li, W. Chen, Z. Cai, and Y. Fang. Caka: a novel certificateless-based cross-domain authenticated key agreement protocol for wireless mesh networks. *Wireless Networks*, 22:2523–2535, 2016.

[4] A. Nanda, P. Nanda, X. He, A. Jamdagni, and D. Puthal. A hybrid encryption technique for secure-glor: The adaptive secure routing protocol for dynamic wireless mesh networks. *Future Generation Computer Systems*, 109:521–530, 2020.

[5] A. Neumann, L. Navarro, and L. Cerdà-Alabern. Enabling individually entrusted routing security for open and decentralized community networks. *Ad Hoc Networks*, 79:20–42, 2018.

[6] A. Sgora, D. D. Vergados, and P. Chatzimisios. A survey on security and privacy issues in wireless mesh networks. *Security and Communication Networks*, 9(13):1877–1889, 2016.

[7] S. Shrestha, M. Andreoni Lopez, M. Baddeley, S. Muhaidat, and J.-P. Giacalone. A time-bound continuous authentication protocol for mesh networking. In *2021 4th International Conference on Advanced Communication Technologies and Networking (CommNet)*, pages 1–6, 2021.

[8] R. Vijayanand, D. Devaraj, and B. Kannapiran. Intrusion detection system for wireless mesh network using multiple support vector machine classifiers with genetic-algorithm-based feature selection. *Computers & Security*, 77:304–314, 2018.