Proceedings Paper:
Yadav, Poonam, Moulds, Anthony and Gillingham, Peter (2025) Enhancing IoT Defenses Against Radio Jamming:Insights from a Thread Testbed Case Study. In: EuroSec'25:Proceedings of the 18th European Workshop on Systems Security. ACM, pp. 18-25.

https://doi.org/10.1145/3722041.3723096

Enhancing IoT Defenses Against Radio Jamming: Insights from a Thread Testbed Case Study

Poonam Yadav, Anthony Moulds and Peter Gillingham
Computer Science Department
University of York, UK
[poonam.yadav,anthony.moulds,peter.gillingham]@york.ac.uk

ABSTRACT

As the Internet of Things (IoT) ecosystem continues to expand, ensuring robust wireless communication in the face of radio jamming attacks has become a critical concern. This paper presents a comprehensive case study on improving IoT resilience to radio jamming using the Thread protocol within a controlled testbed environment. We investigate the vulnerability of Thread-based IoT networks to constant jamming and implement an effective countermeasure to improve network robustness. Our study focuses on the channel hopping method for countermeasure implementation, demonstrating its effectiveness against jamming attacks through detailed experimental results and analysis. This work underscores how jamming and countermeasure systems can be developed and tested on real hardware, fostering further research in the field of network security. The experimental results provide valuable insights into the efficacy of these strategies in mitigating jamming threats.

CCS CONCEPTS

• **Networks** → **Network performance analysis**; **Network experimentation**; *Wireless access points, base stations and infrastructure*; *Wireless mesh networks*; • **Security and privacy** → **Denial-of-service attacks**; **Vulnerability scanners**.

KEYWORDS

Internet of Things (IoT), Thread, Denial-of-Service (DoS), Jamming, Countermeasures, Security

ACM Reference Format:

Poonam Yadav, Anthony Moulds and Peter Gillingham . 2025. Enhancing IoT Defenses Against Radio Jamming: Insights from a Thread Testbed Case Study. In *The 18th European Workshop on Systems Security (EuroSec '25), March 30–April 3, 2025, Rotterdam, Netherlands.* ACM, New York, NY, USA, 9 pages. https://doi.org/10.1145/3722041.3723096

1 INTRODUCTION

According to Statista [17], 29.42 billion IoT devices are projected to be connected globally by 2030. Despite this rapid growth, concerns about security, privacy, trust, and confidentiality remain significant. IoT devices are deployed across various environments to sense and collect data or to control physical systems. Their applications span from smart cities and environmental monitoring—aimed at enhancing the quality of life—to smart homes with features like intelligent lighting, virtual assistants, and security cameras, as well as utility metering for electricity and gas. IoT devices rely heavily on wireless communication, making them susceptible to radio frequency (RF) jamming attacks, which can disrupt network operations, cause Denial-of-Service (DoS) attacks [7], degrade performance, and compromise service availability.

The Thread protocol [2, 18, 22] is a low-power, IPv6-based wireless networking standard designed for IoT and smart home ecosystems, emphasizing interoperability and seamless IP connectivity. Unlike proprietary or hub-and-spoke architectures, Thread enables direct IP communication between devices, eliminating the need for protocol translation layers. Its self-healing mesh network, built on IEEE 802.15.4, enhances scalability and reliability while ensuring no single point of failure. With native IPv6 support, Thread devices can seamlessly integrate with existing IP-based infrastructure, enabling end-to-end connectivity across local and cloud networks. Integrated with Matter, Thread fosters cross-brand compatibility, allowing diverse smart home ecosystems to communicate without vendor lock-in. Its future-proof design positions it as a foundational protocol for secure, scalable, and energy-efficient IoT deployments in residential and industrial environments.

However, Thread does not natively include mechanisms to mitigate jamming attacks, unlike other network technologies such as Bluetooth [15] and 5G NR [16], which employ adaptive frequency and channel hopping techniques. As a result, if network security is a priority, manufacturers of Thread devices must implement additional countermeasures, though this may introduce compatibility risks with standard Thread implementations. With the increasing adoption of the Thread protocol in commercial smart buildings [1] and industrial IoT applications (IIoT), the security and reliability of its implementation and operation becomes more significant; the need for Thread to gain defensive mechanisms against jamming and other forms of DoS is now more urgent.

In this paper, we investigate the specific vulnerability of the Thread network to constant jamming attacks and examine the implementation process of a suitable jamming system. Constant jamming is chosen over disruptive, random, or reactive methods because it is simpler, faster and least expensive to deploy, and consequently more likely to be used by an attacker. The realization of an appropriate countermeasure is then explored. Our key contributions are as follows:

(a) We demonstrate the methodology to establish a Thread network using a custom-built testbed that accurately replicates

EuroSec '25, March 30–April 3, 2025, Rotterdam, Netherlands

ACM ISBN 979-8-4007-1563-1/25/03
https://doi.org/10.1145/3722041.3723096

real-world mesh network conditions. This setup allows for controlled experimentation and reproducibility, serving as a robust platform for security assessments.

(b) We present a systematic procedure for constructing and configuring a jamming system utilizing readily available, off-the-shelf (OTS) devices. This highlights the feasibility of low-cost, real-world jamming threats, thereby emphasizing the need for effective countermeasures in IoT environments.

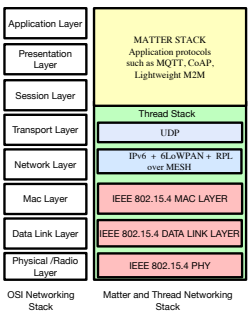
(c) We outline the steps necessary to design and deploy practical countermeasures against jamming attacks directly on Thread-enabled devices. Our approach ensures that the proposed solutions are not just theoretical but applicable in real-world scenarios.

Furthermore, we provide comprehensive experimental results that evaluate the performance of both the jamming system and the implemented countermeasure. Through detailed analysis, we offer insights into the effectiveness of mitigation techniques, contributing valuable knowledge to the security and resilience of Thread-based IoT networks.

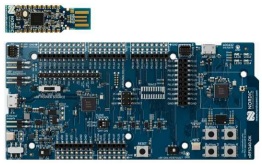
The outline of the paper is as follows: in section 2, we provide background on the IEEE 802.15.4 standard and the Thread protocol; in section 2.3, we provide an understanding of the vulnerabilities and threats to wireless networks through related work; section 3 explains the experimental setup in the laboratory and the design and implementation of our jamming and countermeasure systems; in section 4 we present the results of the experiments and analysis; and finally, in section 5, we present the conclusion and future work.

2 BACKGROUND AND RELATED WORK

This section briefly overviews the IEEE 802.15.4 standard and the Thread and Matter protocols and related work on Radio Jamming and countermeasures. Thread is build on top of the IEEE 802.15.4 standard, offering a complete communication stack for IoT devices.



(a) The protocol stacks.



(b) Two Thread capable devices, nRF52840 USB Dongle alongside the nRF5340DK development board, we used in creating the Thread network testbed.

Figure 1: (a) shows the full Matter-Thread networking stack in relation to the OSI model, while (b) displays the Nordic Semiconductor nRF52840 USB dongle and nRF5340DK development board.

2.1 IEEE 802.15.4

The IEEE 802.15.4 standard [6] enables low-powered, low-bandwidth, and low-cost devices to be networked by defining the physical (PHY) and media access control (MAC) layers in the Open Systems Interconnection (OSI) model, as shown in Fig. 1(a). Devices can connect using either a star or peer-to-peer topology. The MAC layer interfaces the physical and network/application layers. Since IEEE 802.15.4 does not specify the network/application layers, this is implemented by protocols such as Zigbee or Thread/Matter. Additionally, the MAC layer employs carrier-sense multiple access with collision avoidance (CSMA-CA) to prevent data collisions, and offers guaranteed transmission time slots for each device. The MAC handles all data transfer between the physical and network layers. The standard specifies the use of radio for its physical medium; sub-GHz 780/868/915 MHz bands and the 2.38/2.45 GHz bands. For Thread, only the 2.45 GHz band is used.

2.2 Thread and Matter

Thread [18] is designed to create secure, reliable, and scalable mesh networks for connecting and controlling IoT devices. Built on top of a modified sub-set of the IEEE 802.15.4 standard, which defines the physical (PHY) and media access control (MAC) layers, Thread uses 6LoWPAN (IPv6 over Low-Power Wireless Personal Area Networks) to allow each device in the network to have an IP address, see Fig. 1(a). It creates a self-healing mesh network where devices can communicate directly or through other devices, enhancing reliability and range. Thread includes robust security features such as AES-128 encryption for data protection and secure device authentication. Optimized for low power consumption, Thread is ideal for battery-powered devices and is designed to support networks with hundreds of devices, ensuring scalability for large deployments. A Thread network contains two types of nodes - routers, which forward network packets and provide secure commissioning services for new devices, and end devices, which primarily communicate with a single router and do not forward network packets. Nodes within a Thread network are categorized as follows:

(1) **Full Thread Device (FTD)**. These devices can be routers or eligible to be promoted to a router. FTDs always have their radios on and maintain IPv6 address mappings of devices on the network.

(2) **Minimal Thread Device (MTD)**. All messages from these devices are forwarded to their parent router. MTDs can either be Minimal End Devices (MEDs), or Sleepy End Devices (SEDs); SEDs are normally disabled and wake occasionally to poll for messages. MTDs can only function as end devices.

Within a Thread network, specific roles include Thread Leader, which is a router managing other routers, and Border Routers, which forward data between a Thread network and a non-Thread network. Additionally, Thread can upgrade or downgrade routers as needed to ensure the most efficient network operation.

A Thread network can have up to 64 Routers (32 active at any given time), each with 512 children, giving a theoretical maximum of 32768 devices connected at once. Thread can achieve this through utilizing its low data rate of 250 kbps between devices, and the mesh network, through which devices communicate by routing data to the Router they are connected to (in a Parent/Child relationship), which then handles sending the message onto the receiving device.

Matter [3] is an IPv6-based application layer protocol that supports TCP/UDP for data transmission, enabling broad compatibility across networks like Thread and Wi-Fi.

2.3 Security Vulnerability, Jamming and Countermeasures

The research on Thread security vulnerabilities are limited. However, since both Thread and Zigbee are based on IEEE 802.15.4, many security risks affecting Zigbee could also impact Thread networks. Several studies have explored vulnerabilities in Zigbee, particularly in areas such as key sharing, replay attacks, battery depletion, and denial-of-service (DoS) attacks [4, 5, 8, 19, 21]. This highlights the need for further security analysis and countermeasure development tailored specifically for Thread.

While the security of IoT networks against jamming attacks has been extensively explored [23], there remains a critical gap in understanding the real-world effectiveness of countermeasures specifically within Thread-based networks. Existing studies, such as those by Xu et al. [20] Akestoridis et al. [2], Yadav et al.[11, 22] and Liu et al. [9], have thoroughly categorized jamming techniques and highlighted Thread's inherent vulnerabilities to jamming and other radio-based attacks, such as battery depletion and Denial of Service (DoS) attacks.

However, these works primarily focus on identifying vulnerabilities, classifying attack types, and demonstrating the feasibility of attacks without providing comprehensive, experimentally validated countermeasures, particularly for Thread networks. While some studies acknowledge that Thread is vulnerable to jamming, they lack a detailed exploration of practical, hardware-implemented countermeasures tailored to Thread's unique architecture, mesh networking behavior, and energy constraints.

Although battery depletion attacks and CSMA-CA deviations have been demonstrated on real hardware, there is a significant gap in research evaluating defensive mechanisms against these attacks within controlled, Matter/Thread-specific testbeds. Existing studies primarily focus on attack feasibility, leaving mitigation strategies largely unexplored.

To address this, our work not only examines traditional noise-based jamming but also introduces a continuous deceptive packet injection technique, leading to receiver saturation and rendering Thread nodes completely inoperable. This dual-mode attack strategy—combining RF jamming with deceptive packet flooding—is a novel contribution, highlighting multiple previously unexamined attack vectors against Thread networks. By validating these attacks within a controlled testbed, we provide the first comprehensive evaluation of their impact, bridging the gap between theoretical vulnerabilities and real-world network resilience.

2.4 Threat Model

In the threat model, we define Threat Actors, Attack Vector, Attack Feasibility, Impact on Thread Networks, and finally, Mitigation Strategies.

The ***Threat Actors*** A are potential adversaries targeting Thread networks include malicious hackers aiming to disrupt IoT infrastructure, competitors seeking to interfere with enterprise deployments, state-sponsored attackers targeting critical infrastructure, and cybercriminals using jamming as part of broader denial-of-service (DoS) attacks.

The ***Attack Vector*** V is a function that represents how an attack is executed:

$$V : A \rightarrow R$$

where R is the set of radio jamming techniques used by the attacker. Specifically, we define:

$$R = \{r_c, r_p, r_s\}$$

where:

r_c = Continuous jamming
r_p = Pulse or reactive jamming
r_s = Sweep jamming

The effectiveness of an attack vector $V(A_i)$ depends on the jamming power J and the signal-to-noise ratio SNR:

$$V(A_i) = f(J, SNR)$$

where higher J and lower SNR increase the likelihood of network disruption. In our work, we focus solely on continuous jamming (weak adversary).

$$R = \{r_c\}$$

The ***feasibility of an attack*** is determined by:

$$F = g(D, C, E)$$

where:

D = Discoverability of the Thread network
C = Cost of attack execution
E = Ease of attack implementation

An attack is more feasible if D is high (network easily detectable), C is low (low-cost Software Defined Radios (SDRs) available), and E is high (simple jamming methods exist).

The ***impact function*** I quantifies network disruption as a function of attack effectiveness:

$$I = h(V, F, R)$$

where:

V = Attack vector used
F = Feasibility of execution
R = Resilience of the Thread network

We define network degradation D_n as:

$$D_n = \frac{P_j}{P_s}$$

where:

P_j = Power of jamming signal
P_s = Power of legitimate signal

If $D_n > \theta$ (threshold), the network collapses.

Mitigation strategies M aim to reduce I (impact) and increase resilience. This is modeled as:

$$M = k(H, M_d, M_p)$$

where:

H = Adaptive frequency hopping
M_d = RF signal monitoring and detection
M_p = Physical security and network redundancy

The effectiveness of mitigation M is given by:

$$E_M = \frac{I_{\text{without mitigation}} - I_{\text{with mitigation}}}{I_{\text{without mitigation}}}$$

where a higher E_M indicates a more effective countermeasure.

3 EXPERIMENT SETUP

As described earlier, Thread is vulnerable to radio channel jamming. When effective, the jamming acts as a form of DoS attack, preventing the network from operating. Even partial jamming, i.e., where only part of the meshed network is compromised, can have a significant impact, e.g. in security systems relying on sensor devices. In this section, we describe the experiments performed in order to examine this type of network vulnerability and detail a jamming method and its countermeasure.

3.1 Thread Edge Testbed

A bespoke Testbed was constructed in order to perform the experimentation with Thread networking, enabling the easy placement of Thread devices to a wall-mounted acrylic panel. For the experiments in this paper, a Thread network comprising of twenty Thread nodes was created, a Border Router (BR), four FTDs and fifteen MTDs. For the FTD devices, Nordic Semiconductor nRF5340DK development boards [13] were used and for the MTD nodes the Nordic Semiconductor nRF52840 USB Dongle device [12] was chosen, as shown in Fig. 1(b). Each Thread device was connected to a Raspberry Pi 5 SBC (RPi) via its USB port in groups of three MTDs and one FTD per RPi. The connection provided power and a VCOM (UART) link for debugging and log dumps. The BR was formed from a networked RPi and an nRF5340DK. The resulting Testbed layout is shown in Fig. 3. As can be seen, the Thread devices on the Testbed are located in close proximity to each other. In an attempt to emulate a more normal configuration, where network nodes are separated by greater distances, the radio transceiver's transmit output power levels for each Thread device were reduced, while still maintaining a single unpartitioned network. The FTD devices' transmit power was limited to -20 dBm and the MTDs level further reduced to -40 dBm (we note that the radio receiver's sensitivity is specified at -100 dBm for IEEE 802.15.4 reception). These transmit power levels encouraged End Device (Child) nodes to group locally with their nearest Router (Parent), as indicated in Fig.2.

The fifteen MTD Thread devices were programmed to act as temperature sensors, periodically broadcasting their CPU temperatures in 32-byte UDP data packets. All FTD and MTD nodes maintained an internal log of all transmitted and received data packets. At the end of each experiment, all node log files were downloaded for network analysis. The source-code is publicly made available at Systron Lab Github[1].

[1]https://github.com/SystronLab/thread-edge-testbed

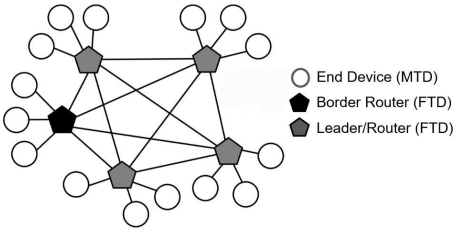


Figure 2: Diagram of the Thread network realized on the Testbed. All FTDs were mesh-linked, with groups of three MTDs (Child nodes) attached to a unique Router (Parent).

Figure 3: Image of the Thread Edge Testbed. In the experiments, twenty Thread devices were mounted. A group of three MTDs and an FTD were connected to an RPi via their USB ports, repeated three times. A fifth RPi was used together with an FTD to form a Border Router. All RPis were networked to an embedded PC via an Ethernet switch.

3.2 Channel Jamming

A DoS attack on a Thread network is achievable by applying a constant jamming signal on the correct network channel. In the experiments, an O-QSPK-250 modulated carrier signal with pseudo-random bit sequence (PRBS) data was initially used. A second approach was to inject a stream of empty IEEE 802.15.4 packets on to the network (a form of deceptive jamming) to throttle or choke the MAC sub-layer. The two methods gave the same jamming performance in our experiments, each requiring only knowledge of the network radio channel (PAN ID and network key were not required). With short jamming packet inter-frame spacing (IFS), typically below 4 ms, the Thread devices on the network were observed to be unable to transmit (due to the CSMA-CA mechanism in their MAC sub-layer), and in addition unable to receive any non-jamming packets due to receiver saturation. In the following experiments, only a single channel is attacked at once.

The jamming attack was performed by using a Sewio OpenSniffer device [14] and directional antenna for the IEEE 802.15.4 packet

injection, and a Nordic Semiconductor nRF5340DK for Thread network scanning; the equipment was controlled by a Python script, linking the devices. The attack system was developed under two scenarios: the first covered detecting a network radio channel and attacking it indefinitely, enabling the attacker to compromise the Thread network with no implemented countermeasures; the second scenario served to counter the defense mechanism described later in section 3.3, with a repeated channel sniffing and attack process, to target any channel that the network under attack may hops to. The jamming system assumes the attacked network is the only Thread network available and then only using a single radio channel.

3.2.1 Channel Sniffing. The Nordic Semiconductor nRF5340DK is used for its network scanning functionality in this implementation. Connecting to the network is not necessary, only knowledge of the channel ID that the network is occupying. The OpenThread CLI makes available the 'scan' command, which scans for available Thread networks using the Mesh Link Establishment (MLE) protocol without knowledge of the network key. This command retrieves the channel ID number, which is necessary for identifying which channel to attack, the Personal Area Network ID (PAN ID), Extended PAN ID and network name; these additional details help identify the discovered network is a valid Thread network. From this, the channel number is retrieved and used as the target for packet injection from the OpenSniffer.

3.2.2 Channel Jamming. The Sewio OpenSniffer device is used for packet injection in this jamming implementation. After obtaining the channel number through the network detection process, the OpenSniffer is instructed to inject IEEE 802.15.4 packets onto the channel with the minimum permitted inter-frame spacing (IFS) of 1 ms. Packet-based jamming is preferred over modulated radio signal interference, as it enables network jamming event marking when using the Wireshark network analyzer. The transmitted packets contain no payload and are replayed without frame count restrictions. The OpenSniffer is controlled via its RJ45 port, receiving HTTP commands with the channel number passed through query parameters. The signal is amplified using a 20 dB wideband LNA and transmitted through a 12 dBi directional antenna, ensuring sufficient power to cover the entire Testbed. Figure 4 illustrates the antenna placement. The antenna transmitted approximately +15 dBm of signal power, resulting in a minimum RSSI value of -48 dBm across the entire Testbed.

3.2.3 Process. Using the nRF5340DK board and the OpenSniffer for channel detection and packet injection, respectively, requires a simple process loop to implement the jamming attack; the Python script is used to control the loop and handle communication between the two devices. The process begins by detecting the channel with the nRF5340DK, sending the 'scan' command through the serial interface and parsing the response to obtain the channel number, PAN ID, external PAN ID, network name and link quality indication (LQI) to determine which channel is to be attacked. In this scenario only one network is available. Once the channel ID number is determined, an HTTP request is made to the OpenSniffer, directing it to inject packets onto the channel. These packets consist IEEE 802.15.4 headers with no data. To combat the channel hopping

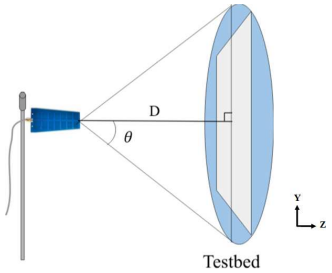


Figure 4: Yagi antenna placed directly in front of the Testbed. Distance D = 1.5 m for complete coverage (θ = 25°). Vertical polarization in Z-Y plane.

countermeasure described later, this process simply needs to run in a loop, detecting the channel repeatedly and waiting for a channel hop. Once a new channel has been detected, a new HTTP request can be sent to the OpenSniffer, which changes its target channel and begins injecting packets on that channel.

3.3 Jamming Countermeasure

When active, the single channel continuous radio jamming apparatus, described in this paper, attacks the IEEE 802.15.4 PHY in the Thread stack layer of each node on the Thread network. If sufficient RF energy is received from the transmitted jamming source on the correct channel by the PHY's receiver, the Thread device's MAC sub-layer inhibits transmitting packets due to a poor clear quality assessment (CCA) result in the CSMA-CA mechanism. In addition, reception of any valid IEEE 802.15.4 packets will be rendered impossible due to useless or severely degraded S/N ratio at the PHY receiver, where the 'useful' signal content is lost in the jamming 'noise'. Consequently, the Thread nodes are incapable of both transmitting and receiving IEEE 802.15.4 packets. Similarly, when attacking with an uninterrupted continuous stream of deceptive packets, the Thread node interface becomes unusable due to receiver saturation. In the case of either jamming sources, de-tuning the PHY's narrow-band transceiver to another channel however will remove the jamming event. Hence, a valid countermeasure is to perform a channel hop to an alternative unoccupied frequency. This section describes an implementation of this countermeasure method on the Testbed. Thread devices were programmed with firmware written in the C programming language, incorporating Google's well-known open-source OpenThread API [10].

3.3.1 Jamming Detection. An essential element in implementing a countermeasure to the continuous jamming system described in this paper is the reliable detection of the jamming event. This was achieved in the Thread device's firmware by monitoring the PHY layer's Received Signal Strength Indicator (RSSI) value. In order to determine a detector threshold, the minimum RSSI value for all Thread nodes was measured across the Testbed during Testbed setup (with no jamming applied) and found to be -48 dBm. In the experiments, the threshold was set to -60 dBm to include -12 dBm margin. If the RSSI value continuously exceeded this threshold over a 10 second interval a jamming event was detected. Once

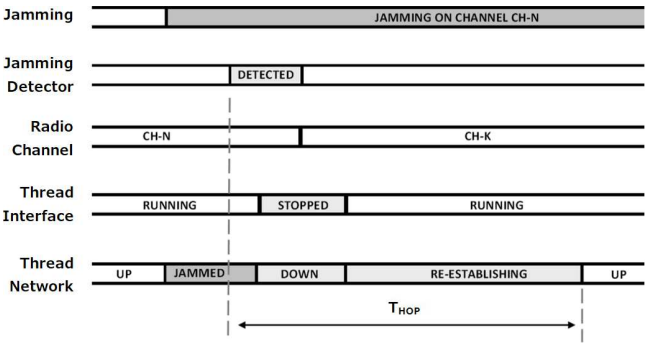


Figure 5: Timing Diagram for the Countermeasure to respond to an active jamming event. The radio channel CH-N is the network channel to be jammed and CH-K the hop channel. T_{HOP} is the time it takes for the Thread network to recover once jamming is detected.

detected, the device would signal the start of a channel hop. The long detection window was set to prevent potential false positives.

3.3.2 Channel Hopping. After detecting a jamming event, each Thread node must prepare to jump to an alternative channel. In the experiment, an array of up to sixteen IEEE 802.15.4 channels was created by the Thread Leader on initial creation of its network (without jamming applied); a frequency scan was performed to determine the available unoccupied channels with the found channel IDs stored in the array. After sorting, the array contents were ordered randomly to form a Hop List, with a Hop Index created to point to the next hop in the list. For the experiments, the Hop List created by the Leader was as follows:

15 12 17 20 14 11 18 25 24 13 19 21 16 22 26 23.

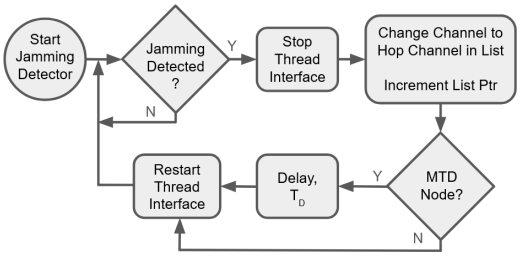


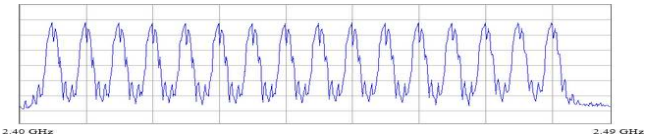
Figure 6: Flow diagram of the implemented Countermeasure.

As Thread devices joined the experimental network, the Hop List and Hop Index were disseminated across all nodes; the Routers fetched the list and index from the Leader, while Child devices obtained the information from their Parent. At this point, all nodes were in sync and prepared to hop to the agreed alternative channel if a jamming attack occurred. Fig. 6 gives a flow diagram for the implemented Countermeasure. To limit unintentional side-effects on the network when changing channel frequency, each node first disabled its Thread interface using OpenThread API calls and re-enabled it once its new channel is set. This step ensured that the Thread devices correctly detached and re-attached themselves from and to the network. In Fig. 5, the relative timing of the channel hop mechanism when active jamming occurs is shown; T_{HOP} is the time it takes for the Thread network to perform a channel hop. Eqn. 1 gives the total time for the Countermeasure to respond, from the start of the applied jamming to network recovery; T_{JD} is the jamming detector delay. For a Thread network topology of just one FTD and N number of reachable MTD devices, Eqn. 2 gives the hop time; T_C is the short interval taken to apply the channel change, T_L is the time the Thread Leader requires to restart its network and T_{MTD} is the time the MTD device take to re-attach to the network. Eqn. 3 applies to the more typical case where there are multiple FTD devices in the Thread network. Here, an optional fixed delay T_D is added for all MTD nodes before attempting to re-attach. The added delay must be greater than the time it takes for the remaining FTD devices to re-join the network, T_R. Though not strictly necessary, delaying the MTDs in this way helps the network topology to remain unaltered. Minimal End Devices and Sleepy End Devices (both are types of MTD) used the same mechanism to perform the channel hop.

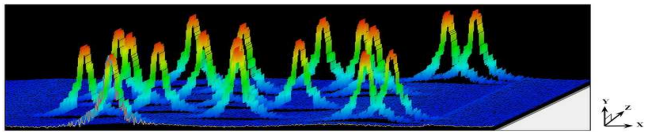
$$T_{CM} = T_{JD} + T_{HOP} \tag{1}$$

$$T_{HOP} = T_C + T_L + max\{T_{MTD_1}, ...T_{MTD_N}\} \tag{2}$$

$$T_{HOP} = T_C + T_L + max\{T_{D_1}, ...T_{D_N}\} \quad \text{when } T_D > T_R \tag{3}$$



(a) Frequency Spectrum with Peak Hold (from 2.40 GHz, span 90 MHz, 6dB/div)



(b) 3D Frequency Spectrogram (X,Y,Z planes for Frequency, Power, and Time respectively)

Figure 7: Measurements using a Tektronix RSA306A spectrum analyzer, showing the transmitted signals from the Jamming apparatus. (a) displays the overlaid captured frequency spectrum for all sixteen transmitted (jammed) IEEE 802.15.4 channels; (b) presents the same captured signals showing their timing relationship during Countermeasure channel hopping (ch26 is shown at the rear with the next hop at ch23, and similarly for other channels based on the Hop List).

4 RESULTS AND ANALYSIS

This section of the paper presents results of the experiments performed using the bespoke Testbed, and includes the evaluation and analysis of the developed Jamming apparatus and the performance of the Countermeasure implementation. We also discuss its limitations and how to deal with the case of perpetual jamming.

4.1 Jamming Effectiveness

The jamming attack can be evaluated in two parts, network detection and packet injection, with both combining to successfully discover and jam a Thread network. The network detection, achieved through the use of a single nRF5340DK board, successfully detected experimental Thread networks across all sixteen channels. The detection implementation was simple to setup and deploy, issuing OpenThread CLI commands via a Python script. With the extraction of the channel ID number via the nRF5340DK, continuous back-to-back packet injection successfully occurred through issuing HTTP request made to the OpenSniffer, again through the Python script controlling the attack.

The success of the jamming system is demonstrated in Fig. 7, showing the jamming of all sixteen channels in succession while tracking the countermeasure; (a) shows the transmitted output at each channel frequency over the duration of the entire attack scenario, with channels being successfully occupied by the jamming signal; (b) shows the attack through time, demonstrating the successful channel switching, once the re-established Thread network has been detected.

With the assumption that the Thread network under attack is the only network available (and not partitioned) and all nodes are in range of the jamming system, the attack would successfully collapses a Thread network, and in the second described scenario, would continue to do so as the channel hopping countermeasure takes action.

4.2 Countermeasure Performance

Two experiments were completed on the Testbed to evaluate the performance of the Countermeasure in increasing Thread network complexity; the first executed on a network topology comprising of just one FTD and up to nineteen MTDs, and the second comprising five FTDs and fifteen MTDs. Using the timing information contained in the on-device timestamped data and event logs maintained in each Testbed Thread device, it was shown that each node detected the jamming event and successfully performed a channel hop. Fig. 9 gives the results for the first experiment. As can be seen in the graph, the T_{HOP} value increases as more Thread devices exist in the network; this is believed to due to normal Thread network behavior and not due to the implementation of the Countermeasure. The averaged T_{HOP} value for FTD devices in the second experiment was determined to be 29 seconds over the sixteen hops. The jamming detector window set at ten seconds in the experiments proved to be sufficiently long to prevent false positives, i.e. normal Thread network traffic failed to trigger a channel hop in any network node.

The Countermeasure has been shown to continuously apply its sequence of channel hops during active tracking by the Jamming system. The jammer has a finite turnaround delay as it must re-scan the network, as illustrated in Fig. 8. If this delay is sufficiently long, the Border Router or Leader would be able to issue network-wide messages, potentially issuing instructions to mitigate further attacks.

4.3 Assumptions and Limitations

The jamming system used in the experiments was, by design, limited to jamming a single radio channel at a time. However, it was

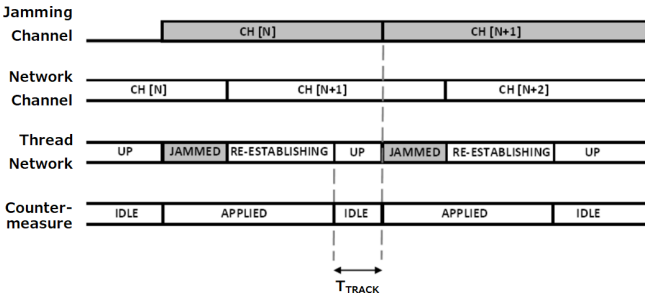


Figure 8: The Countermeasure must continuously respond to active jamming, where the jammer is capable of tracking the network channel hops. T_{TRACK} is the jamming turn-a-round delay.

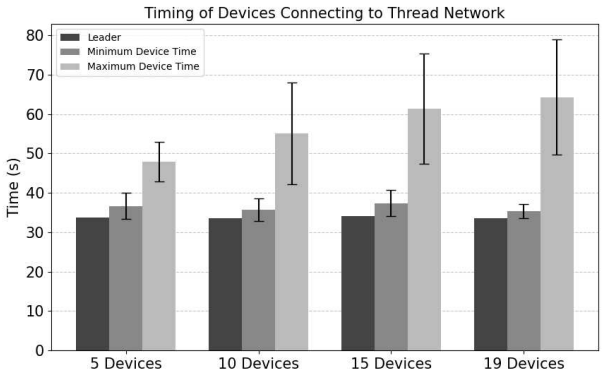


Figure 9: Chart showing the minimum and maximum Countermeasure hop timings for the Thread network comprising of a single FTD (Leader) and up to nineteen MTDs.

sufficiently sophisticated to enable automatic tracking of network channel hops and subsequently re-applying its attack. The system assumed only one Thread network existed and not partitioned.

The Countermeasure suffered from long latencies due to its wide jamming detector window, and the necessity to restart the Thread network; especially with the additional (optional) MTD attachment hold-off delay. Only sequential single channel jamming events could be defended against using the Countermeasure. In addition, an attack using a powerful wideband jamming signal on the Testbed would certainly succeed and difficult to defend against.

The Countermeasure performed well when jamming was applied to the entire Thread network, where each Thread device detected the same jamming event and subsequently all nodes applied their channel hop in sync. If, however, a network node was unable to detect jamming (possibly by being out of range of the jammer), its radio channel would become out-of-step with the sequenced hopped network channel. Though not tested, this scenario could be resolved by the device detecting its detached state from the network and attempt to re-attach by looping through the channels in the shared channel Hop List; once rejoined, it would be in sync with the other Thread devices. Importantly, however, unjammed FTD

devices would need to ensure they gracefully re-attach, avoiding creating a new network on an old/false channel.

4.4 Jamming Attack Mitigation

The experimental results show that it was possible to repeatedly detect radio jamming of the Thread network and that a Countermeasure, by applying channel hopping, proved reliable. However, if the Jamming equipment is continuously tracking the network, then the networked system is perpetually compromised. The network's Border Router(s) can be configured to broadcast an alert message (via Ethernet, cellular network, LoRaWAN, etc.) flagging the DoS attack, so action can be taken. In addition, the individual Thread devices can be programmed or configured to enter a 'safe' mode during the attack period. For example, a door lock should close when attacked or a security lamp should illuminate.

5 CONCLUSION AND FUTURE WORK

The Thread protocol relies on radio communication to establish network links between IoT devices, making it inherently vulnerable to radio jamming attacks. Through a bespoke testbed, we systematically examined these vulnerabilities and demonstrated how relatively simple it is to jam or disrupt radio links, effectively executing a network-wide denial-of-service (DoS) attack. In real-world home or commercial Thread/Matter networks, our results indicate that targeting the Border Router—which typically acts as the single orchestrator for all linked devices—would be sufficient to cripple the entire network if the attacker possesses adequate transmitter power. Furthermore, our findings suggest that expanding the jamming system to disrupt multiple radio channels simultaneously could significantly increase the attack's effectiveness.

To address this vulnerability, we implemented a channel hopping countermeasure on off-the-shelf Thread-capable hardware, demonstrating its effectiveness against active jamming attacks. Our bespoke testbed allowed us to validate that Thread devices can dynamically switch frequencies, maintaining network availability under hostile conditions. The countermeasure operates continuously, iterating through an available channel list until either the jammer ceases tracking the network or the Thread system enters a fallback mode.

This paper not only identifies critical vulnerabilities in Thread networks but also proposes and implements a practical, real-world countermeasure to mitigate these threats. By leveraging the OpenThread API in a controlled testbed, we experimentally demonstrate how frequency agility can restore network functionality even under active interference. This moves beyond theoretical discussions, offering an actionable, deployable defense strategy that Thread device manufacturers and network operators can integrate to enhance resilience against targeted jamming attacks.

Further improvements to the countermeasure will focus on reducing channel hop latency, handling partial jamming attacks more effectively, and developing capabilities to counter multi-channel jamming, ensuring greater robustness in future deployments.

6 ACKNOWLEDGMENTS

The authors would like to thank Mr Dylan Thompson for his contribution in helping construct the Thread Testbed. The research is funded by EPSRC & DSIT funded projects EP/X040518/1, EP/Y037421/1, and EP/Y019229/1.

REFERENCES

[1] 2019. Thread 1.2 in Commercial White Paper. https://www.threadgroup.org/Portals/0/documents/support/ThreadInCommercialWhitePaper_2542_1.pdf. [Online, accessed October 2024].

[2] Dimitrios-Georgios Akestoridis, Vyas Sekar, and Patrick Tague. 2022. On the Security of Thread Networks: Experimentation with OpenThread-Enabled Devices. In *Proceedings of the 15th ACM Conference on Security and Privacy in Wireless and Mobile Networks* (San Antonio, TX, USA) *(WiSec '22)*. Association for Computing Machinery, New York, NY, USA, 233–244. https://doi.org/10.1145/3507657.3528544

[3] Connectivity Standards Alliance. 2014. Matter. https://csa-iot.org/all-solutions/matter/. Accessed on 1st March 2023..

[4] X. Fan, F. Susan, W. Long, and L. Shangyan. [n. d.]. Security analysis of Zigbee. https://courses.csail.mit.edu/6.857/2017/project/17.pdf. Accessed on 30th Jan 2023, publication date: 18th May, 2017.

[5] Angelo Feraudo, Diana Andreea Popescu, Poonam Yadav, Richard Mortier, and Paolo Bellavista. 2024. Mitigating IoT Botnet DDos Attacks through MUD and eBPF based Traffic Filtering. In *25th International Conference on Distributed Computing and Networking (ICDCN)*. 1–12. https://doi.org/abs/2305.02186

[6] IEEE Std 802.15.4 Working Group. 2020. IEEE Standard for Low-Rate Wireless Networks. *IEEE Std 802.15.4-2020 (Revision of IEEE Std 802.15.4-2015)* (2020), 1–800. https://doi.org/10.1109/IEEESTD.2020.9144691

[7] Prabhakaran Kasinathan, Claudio Pastrone, Maurizio A. Spirito, and Mark Vinkovits. 2013. Denial-of-Service detection in 6LoWPAN based Internet of Things. In *2013 IEEE 9th International Conference on Wireless and Mobile Computing, Networking and Communications (WiMob)*. 600–607. https://doi.org/10.1109/WiMOB.2013.6673419

[8] Salam Khanji, Farkhund Iqbal, and Patrick Hung. 2019. ZigBee Security Vulnerabilities: Exploration and Evaluating. In *2019 10th International Conference on Information and Communication Systems (ICICS)*. 52–57. https://doi.org/10.1109/IACS.2019.8809115

[9] Yu Liu, Zhibo Pang, György Dán, Dapeng Lan, and Shaofang Gong. 2018. A Taxonomy for the Security Assessment of IP-Based Building Automation Systems: The Case of Thread. *IEEE Transactions on Industrial Informatics* 14, 9 (2018), 4113–4123. https://doi.org/10.1109/TII.2018.2844955

[10] OpenThread. . API. https://openthread.io/reference/. Accessed on 7th November 2024.

[11] Anthony Moulds Poonam Yadav and Peter Gillingham. 2025. Poster: Evaluation of Radio Jamming Countermeasures in IoT Thread Networks. In *Network and Distributed System Security (NDSS) Symposium* (USA) *(NDSS'2025)*.

[12] Nordic semiconductor. . nRF52840 USB Dongle. https://www.nordicsemi.com/Products/Development-hardware/nrf52840-dongle/. Accessed on 7th Nov 2024.

[13] Nordic semiconductor. . nRF5340-DK Development Board. https://www.nordicsemi.com/Products/Development-hardware/nRF5340-DK/. Accessed on 7th Nov 2024.

[14] Sewio. [n. d.]. Open Sniffer. https://www.sewio.net/open-sniffer/. Accessed on 23rd September 2023.

[15] Bluetooth SIG. [n. d.]. Blutooth Technology Overview. https://www.bluetooth.com/learn-about-bluetooth/tech-overview/. Accessed on 7th Feb 2025.

[16] 3GPP Mobile Broadband Standards. [n. d.]. 5G System Overview. https://www.3gpp.org/technologies/5g-system-overview/. Accessed on 7th Feb 2025.

[17] Statista. [n. d.]. Number of internet of things (iot) connected devices worldwide from 2019 to 2021, with forecasts from 2022 to 2030, 2022. https://www.statista.com/statistics/1183457/iotconnected- devices-worldwide/.

[18] Threadgroup. . Overview of Thread. https://www.threadgroup.org/What-is-Thread/Overview/. Accessed on 25th Oct 2022.

[19] Mohammad Shafeul Wara and Qiaoyan Yu. 2020. New Replay Attacks on ZigBee Devices for Internet-of-Things (IoT) Applications. In *2020 IEEE International Conference on Embedded Software and Systems (ICESS)*. 1–6. https://doi.org/10.1109/ICESS49830.2020.9301593

[20] Wenyuan Xu, Wade Trappe, Yanyong Zhang, and Timothy Wood. 2005. The feasibility of launching and detecting jamming attacks in wireless networks. In *Proceedings of the 6th ACM International Symposium on Mobile Ad Hoc Networking and Computing* (Urbana-Champaign, IL, USA) *(MobiHoc '05)*. Association for Computing Machinery, New York, NY, USA, 46–57. https://doi.org/10.1145/1062689.1062697

[21] Poonam Yadav, Qi Li, Richard Mortier, and Anthony Brown. 2019. Network Service Dependencies in Commodity Internet-of-things Devices. In *Proceedings of the International Conference on Internet of Things Design and Implementation* (Montreal, Quebec, Canada) *(IoTDI '19)*. ACM, New York, NY, USA, 202–212. https://doi.org/10.1145/3302505.3310082

[22] Poonam Yadav, Nirdesh Sagathia, and Dan Wade. 2024. Demo: Battery Depletion Attack Through Packet Injection on IoT Thread Mesh Network. In *2024 16th International Conference on COMmunication Systems & NETworkS (COMSNETS)*. 318–320. https://doi.org/10.1109/COMSNETS59351.2024.10426916

[23] Bo Yang. 2020. A highly-random hopping sequence for jamming-resilient channel rendezvous in distributed cognitive radio networks. *Computers Security* 96 (2020), 101809. https://doi.org/10.1016/j.cose.2020.101809