Wireless Latency Shift Keying

Jacob Johnson, Cody Arvonen, Ashton Palacios, Philip Lundrigan
{jacobej,codyarvonen,apal6981,lundrigan}@byu.edu
Brigham Young University
Provo, UT, USA

ABSTRACT

IEEE 802.11 (WiFi) only has two modes of trust—complete trust or complete untrust. The lack of nuance leaves no room for sensors that a user does not fully trust but wants to connect to their network, such as a WiFi sensor. Solutions exist, but they require advanced knowledge of network administration. We solve this problem by introducing a new way of modulating data in the latency of the network, called Latency Shift Keying. We use specific characteristics of the WiFi protocol to carefully control the latency of just one device on the network. We build a transmitter, receiver, and modulation scheme that is designed to encode data in the latency of a network. We develop an application, Wicket, that solves the WiFi trust issue using Latency Shift Keying to create a new security association between an untrusted WiFi sensor and a wired device on the trusted network. We evaluate its performance and show that it works in many network conditions and environments.

CCS CONCEPTS

• **Networks** → **Network security**; *Cross-layer protocols*; • **Computer systems organization** → *Sensor networks.*

KEYWORDS

wireless subprotocol, IoT, sensor networks, 802.11, WiFi

ACM Reference Format:
Jacob Johnson, Cody Arvonen, Ashton Palacios, Philip Lundrigan. 2024. Wireless Latency Shift Keying. In *International Conference On Mobile Computing And Networking (ACM MobiCom '24), September 30–October 4, 2024, Washington D.C., DC, USA.* ACM, New York, NY, USA, 15 pages. https://doi.org/10.1145/3636534.3649373

ACM MobiCom '24, September 30–October 4, 2024, Washington D.C., DC, USA

ACM ISBN 979-8-4007-0489-5/24/09…$15.00
https://doi.org/10.1145/3636534.3649373

1 INTRODUCTION

Low-cost Internet of Things (IoT) devices are plagued with security issues, especially in homes where best security practices are not always followed. IoT devices have been used as backdoors into private networks [9, 10] and utilized as botnets in massive DDOS attacks [25]. As the number of IoT devices deployed increases, as they are forecasted to [14], this problem will grow even worse.

The exploits are possible in part because of the design of WiFi's security. Traditionally, when you connect a device to your WiFi network by providing your network name and password, you give the device full access to your network. One insecure device can compromise a whole network [16]. WiFi has only two modes of trust: complete trust or complete untrust. *What if a user wants to have partial trust of the device?* Many IoT sensors, such as temperature sensors or air quality sensors, do not require all the capabilities of WiFi and should not get full access to a WiFi network to periodically send small amounts of data. This coarse grain access control is akin to having a stranger come to your house and having only two options: not answering the door or answering the door and giving the person the keys to your house. Regrettably, partial trust or partial communication is not possible with standard WiFi, and this is where our work innovates.

This paper introduces a novel way of communicating between *unassociated* WiFi devices and a *trusted* WiFi network without connecting the unassociated WiFi devices to the network. Our solution requires no additional hardware or changes to existing hardware—it is implemented purely in software and uses standard WiFi. Our method creates an air gap for safety between an untrusted IoT sensor and a secured network allowing communication to only go one direction and only when the trusted WiFi network needs to receive data. This method drastically limits the ability of a compromised WiFi device to affect the secured network. Our solution functions as a wicket gate [3], as shown in Figure 1. A wicket gate is a smaller door incorporated into a larger city or castle gate used in the Middle Ages. It allows for more fine-grain passage through the gates. Rather than opening the large and heavy main gate to let a few travelers in, leaving a community vulnerable to potential attacks while the gate is open, the wicket door can be used. Our solution provides similar benefits. It adds a secondary form of communication

Figure 1: An example of a wicket gate [32], to the left of the main gate, used in the Middle Ages to provide fine-grain access to city or castle gates.

to WiFi that is designed for IoT sensor readings that have a different security association than normal WiFi.

Other solutions exist, such as network partitioning using separate WiFi networks (discussed in more detail in Section 2), providing some of the same benefits. However, these solutions have significant drawbacks. They require additional hardware or advanced network configuration of a WiFi network and advanced knowledge of network operation, which makes it challenging to deploy to an application for a typical consumer. Our solution innovates by requiring no additional hardware while still utilizing the main WiFi network, making it very easy to use. A user of our system does not have to configure or adjust anything about their network. Our system is essentially an overlay on top of their existing network.

Our work forms a new communication channel between an untrusted/unassociated WiFi device and a device on the trusted network. The device on the trusted network can be wireless or wired, but for illustrative purposes, we will assume it is wired. On this new communication channel, the untrusted device can send data to the device on the secure and trusted network. However, direct communication between an unassociated WiFi device and a device on a trusted network should not be possible because the communication is encrypted. The critical insight into making this communication channel is that though the *unassociated device can not directly send data to the private network, it can still affect the wireless environment.* A wired device on the private network can not directly receive data from the untrusted device, but it can *detect changes in the wireless environment.* We achieve communication by having the untrusted device strategically and surgically jam the WiFi communication channel, causing the latency of *just one device* to increase momentarily. The pattern in which the unassociated device jams the network conveys information. A device on the trusted network detects the changes in network latency and receives the data. We call this method **Latency Shift Keying** (LSK). LSK provides a new communication channel that can be set up between a device in the trusted network and an untrusted/unassociated WiFi device.

This paper presents a novel form of communication through changes in latency that allows for data to be sent by a device without fully trusting the device. Our new communication channel allows someone to set up a WiFi sensor and have it send data without giving that device full access to the WiFi network. Specifically, the major contributions of this paper are:

- We design a wireless "subprotocol" called **latency shift keying (LSK)** that uses changes in latency to send data on a network. LSK is implemented only in software and requires no changes to WiFi hardware. LSK allows communication between a wireless and a wired device (or two wireless devices), even through a secure WiFi network. LSK is a general-purpose modulation scheme that can be used to transmit arbitrary data.
- We design an application called **Wicket**, that uses LSK to allow a WiFi IoT sensor to send data to a wired entity on a secure network. Wicket provides a bridge between the untrusted IoT sensor and the trusted network. Since LSK offers its own communication channel outside of WiFi while still using WiFi, Wicket creates new security associations between unassociated devices and devices on the trusted network.
- We implement Wicket and LSK on commodity hardware and characterize their performance. We demonstrate the use of Wicket in a variety of environments. We show that Wicket is scalable and robust to other network traffic, and has little impact on network performance. We open-source our protocol on GitHub [22].

Our system constitutes a drop-in replacement for current IoT WiFi sensors, allowing data to be sent without completely trusting the sensor, fixing a fundamental problem with WiFi's security trust model.

2 MOTIVATION

Users want to connect an IoT sensor to their home network without the risk of that device getting compromised and compromising their network. This is at odds with WiFi's security association which gives a device complete access

to the network once connected. LSK and Wicket are geared toward devices that send little amounts of data, such as a environmental sensor. It is designed to be used in applications where receiving data from an unassociated WiFi sensor is worth the increased cost of energy to transmit and load on the network. An example of this use case would be the deployment of WiFi sensors into homes for research studies. In this case the sensor deployment is temporary, and the ability to receive data without permanently adding the device to the network would be beneficial. Furthermore, the sensor may be powered externally via wall power, making the additional energy cost to send the data acceptable.

Approaches exist that allow network managers to partition untrusted IoT sensors into their own network [7]. However, these methods have serious drawbacks that make them unfeasible for typical residential consumers to adopt. In this section, we discuss a commonly used approach to solve the IoT device trust problem and how this approach compares to our proposed solution. We discuss other research related to LSK in Section 7.

The conventional way of preventing an untrusted device from compromising a network is to create two networks: one network for untrusted devices and another for trusted devices. A network manager sets up a WiFi network with two advertised SSIDs. Traffic coming from the different SSIDs are tagged with separate VLAN tags, preventing traffic from mixing. The separate VLANs prevent untrusted devices from accessing the trusted devices. Even if an untrusted device does get compromised, it will only be able to affect the devices on its same network, limiting the scope of an attack.

The major drawback to the above approach is that it requires a network setup that is not feasible for a typical person. Some consumer-grade WiFi equipment cannot broadcast multiple SSIDs and most cannot perform VLAN tagging. Even if the hardware is capable, it requires advanced networking knowledge to set up and maintain. Since applications on a network typically do not have access to modify the configuration of the router, it is difficult to abstract away this complexity from the user through automation. Using this approach takes additional effort and configuration to prevent devices from accessing the Internet; otherwise, the device is still vulnerable to attacks. Even with the untrusted devices partitioned to a separate network, a device can still be used as a botnet node in a DDOS [20]. Also, this solution is error-prone. If a user forgets to use the special untrusted network to connect a device, then the whole network can be compromised. Lastly, researchers have shown that monitoring wireless signals can determine behavior in a home [27, 36, 37]. If a compromised device is allowed to connect to the Internet, it can still cause damage or invade privacy, even if it cannot access your protected network. While all of these problems can be overcome through network configuration, they can not be automated and require much work and configuration from the user.

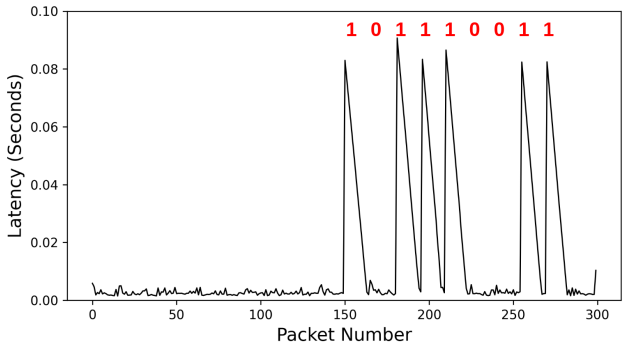

Figure 2: An example of encoding data into the latency of a wireless network.

Our approach does not suffer from the same problems. It creates a communication channel on top of the existing WiFi network, requiring little setup or hardware. A key innovation is that our solution is a software application and does not require direct access to the networking stack or the ability to modify access point configuration. As a result, our solution excels in deployability, meaning that it can be easily automated by the user applications. Each untrusted device creates a unique communication channel with a node on the trusted network. The communication channel allows each device to set up its own security association with the node on the trusted network, partitioning each untrusted device into its own virtual network. Our solution essentially creates an air gap between the untrusted device and the rest of the network, preventing the device from connecting to the Internet or communicating with the network, unless explicitly allowed by the network.

A user could also invest in alternative wireless protocols designed for IoT sensors, such as ZigBee or ZWave. These solutions require a hub that bridges from the wireless network to the home's IP network. This solution can provide isolation for the IoT devices at the cost of requiring extra hardware. Our solution, on the other hand, requires no extra hardware and leverages the massive popularity and low-cost hardware of WiFi. Our design also allows for flexibility. If a user trusts the device or is not worried about being compromised, then they can still connect the WiFi device to their network using normal means. Wicket does not preclude people from using the device as a standard WiFi sensor.

3 LATENCY SHIFT KEYING

The key insight into LSK is that though an untrusted or unassociated device can not directly send data to a trusted WiFi network because of WiFi's security model, it can *still*

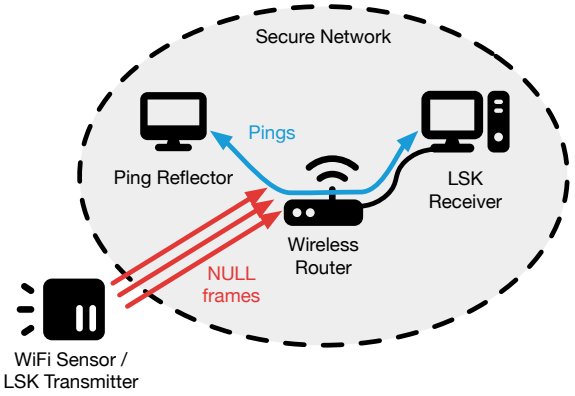

Figure 3: Overview of the Latency Shift Keying Components.

impact the wireless environment of that trusted network. This is because WiFi is broadcast in nature and though a WiFi network might have separated networks, devices still share the underlying RF medium. By having the untrusted device change the environment in a predictable way to send data, a participating device on the trusted network can monitor the environment and decode the data, even from a wired connection. In the case of LSK, we use latency as the way to measure the impact on the network. An example of our approach is shown in Figure 2. Spikes in latency, caused by the transmitter, convey a one and normal latency conveys a zero, as shown in red letters. This method side-steps conventional WiFi security, creating a new side channel of communication outside of the scope of WiFi security while *still using standard WiFi hardware*. We use this novel communication channel to create a new network for untrusted IoT WiFi devices where a device can send data to the network without giving the device access to the WiFi network, which we call Wicket. We acknowledge that this approach fits best with devices that only need to send small amounts of data occasionally, however, this represents a large group of devices types and use cases.

In this section, we describe the details of the general-purpose modulation scheme, LSK. In the next section, we describe the details of Wicket, the application that uses LSK to provide network segregation between trusted and untrusted WiFi devices.

LSK requires three components: a transmitter, a receiver, and a ping reflector. These entities are shown in Figure 3. The LSK transmitter is a WiFi sensor that is not fully trusted to connect to the WiFi network. It deterministically induces latency on one device on a WiFi network to encode its data. The LSK receiver is a wired device on the trusted network. It monitors the latency of the network, looking for encoded messages in the latency. As stated earlier, the LSK receiver can be wireless, but to demonstrate the versatility of LSK, we assume the LSK receiver is wired. This demonstrates LSK's ability to work between two different domains, Ethernet and WiFi. Finally, the ping reflector is a device that is used by the receiver to measure the latency of the network. This device needs to be wireless and respond to ping packets. Otherwise, the ping reflector is oblivious to the protocol. LSK is implemented purely in software, so it does not require modifications to a device's hardware or firmware. The following sections describe the design of the transmitter, receiver, ping reflector, and how the data is encoded in the latency.

3.1 Transmitter

At a high level, the transmitter needs to cause an increase in latency on the network in a specific timed pattern that the receiver can measure. To achieve this, our system takes advantage of the insight that *an untrusted device can cause latency on a trusted network, even while not being part of the network*. There are many ways for an untrusted device to add latency to a WiFi network. The simple act of transmitting on the same channel as the WiFi network can cause a slight delay on the network. However, for this approach to work well, the delay needs to be more pronounced and consistent. We consider three methods for increasing the latency of a network: 1) naive jamming, 2) using 802.11 clear-to-send (CTS) frames, and 3) using 802.11 NULL frames.

3.1.1 Naive Jamming. Naive jamming relies on WiFi's carrier sensing multiple access (CSMA) protocol to induce latency on a network. By having a jamming device transmit a frame, the target network must wait for the transmission to be over plus a random back-off. If the jamming device sends enough frames, competing for airtime with the nodes on the wireless network, it will have a measurable impact on the network's latency. While this approach is the simplest, it has two major drawbacks. First, it requires a constant stream of transmissions by the transmitter to make a noticeable impact on the network. This increases the burden on the transmitter. Second, naive jamming slows down the whole network. Slowing down the whole network is a big price to pay for allowing one device to transmit data. While it is feasible to implement LSK using this approach, it is not very practical.

3.1.2 Clear-to-send (CTS) Frames. Request-to-send (RTS) and clear-to-send (CTS) frames are used by a WiFi station and an access point (AP) to get exclusive access to the channel to transmit [17]. The typical procedure goes as follows. A WiFi station that wants to transmit sends an RTS frame to an AP. To grant exclusive access, the AP transmits a CTS in response. All nodes on the same channel as the network that receive this transaction must not transmit on the channel

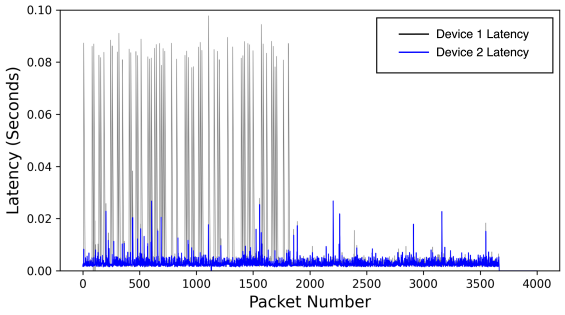

Figure 4: Round trip time for Nodes 1 and 2, with NULL frames directed at jamming Node 1. Node 2 is unaffected by the NULL frames.

for the duration specified in the RTS/CTS frame. For our context, the LSK transmitter can inject an RTS frame into the network even when it is not associated with the network, asking the AP for access. The AP will broadcast a CTS to all nodes within range of the AP. We considered having the transmitter inject its own CTS into the network, but by using an RTS and involving the AP, we are able to increase the range of the CTS transmission. This method has been used by others to clear the channel [23]. This method compared to the naive approach, reduces the burden on the transmitter requiring only one transmission to clear the channel for a certain amount of time. The duration field of the CTS is limited to 32.767 ms [2], so the effect a CTS frame can have is limited. This approach also has the same drawback as the naive approach: a CTS frame negatively impacts the whole wireless network, slowing all nodes.

3.1.3 NULL Frames. NULL frames are a special type of data frame described in the IEEE 802.11 standard [2] with an empty payload. One of their commonly used purposes is to notify an AP that a WiFi station is going into power saving mode [17]. The AP will buffer the frames that it receives on behalf of the device until the device wakes up and sends another NULL frame to indicate to the AP it has woken up. As part of this process, the AP advertises what stations it has buffered frames for in its beacon frame. Since the frame contains no payload and 802.11 headers are not encrypted, it is easy for another device to spoof a NULL frame [18].

Using NULL frames, the transmitter can inject a spoofed NULL frame targeting a legitimate device on the network. The AP will start buffering the frames for that device. The key insight into using a NULL frame, as opposed to a different technique, is that a NULL frame only affects the latency of one device. To show this is the case, we perform an experiment where we ping two devices while sending NULL frames targeting one of the devices. The results are shown in Figure 4. Node 1 is the gray line and node 2 is the blue line. We send NULL frames targeting node 1 while pinging both nodes 1 and 2. The latency of node 1 is severely affected by the NULL frames, showing high spikes in latency as a result of the NULL frames, whereas the latency of node 2 is unchanged. Around packet number 1750, we stop sending NULL frames to node 1, at which time, the latency follows the same pattern as node 2.

Using NULL frames minimizes the impact LSK has on other traffic on the network because it only affects the one device the transmitter is spoofing NULL frames for, in this case, the ping reflector. We can jam one device independently of any other device on the network. Using this ability, we can create multiple orthogonal communication channels by jamming different devices on the network. For example, multiple transmitters can target different ping reflector devices and create multiple parallel LSK streams of data. We discuss multiple transmitters in more detail in Section 4.3.

The transmitter encodes data in the network's latency by strategically injecting NULL frames that target the ping reflector. However, even with using NULL frames and all of the benefits this frame provides, they are not sufficient to *reliably* encoding data in the latency of the network. The system must overcome the problem of premature "unjamming" of the ping reflector.

3.1.4 Premature Unjamming. As part of the 802.11 specification, APs send out beacon frames every 102.4 ms. As mentioned previously, one of the fields of the beacon frame is a list of all of the stations the AP has buffered frames for. While a transmitter is encoding its data in the latency by injecting NULL frames, the ping reflector might receive a beacon frame, because it is not actually in power-saving mode. The ping reflector will see that the AP has buffered frames for it, and request the buffered packets from the AP, undoing the transmitter's jam prematurely. This leads to unpredictable spikes in latency since sometimes the jamming will terminate early.

To address this challenge, we embrace beacon frames and use them as a synchronization mechanism. Before sending NULL frames, the transmitter enters monitor mode [4], which lets it sniff the WiFi frames of the secure network. As soon as it detects a beacon frame, it sends out a NULL frame, maximizing its ability to jam the device. Before the end of the beacon interval, the transmitter sends a NULL frame to the AP requesting packets to be sent to the ping reflector. Using the beacon frame to synchronize prevents potential issues with clock drift between the transmitter and the receiver. We use the time between beacon frames as the basic building block to encode data into the latency. As shown in Figure 5, we encode one peak between two beacon frames. We give more details about how we encode the data in Section 3.4.

3.2 Receiver

To receive the data sent by the transmitter, the receiver must be able to measure the impact of the transmitter on the wireless environment. To do so, the receiver picks a wireless node on the network, called the ping reflector, to measure latency against. It is important that the ping reflector is wireless so that the latency of the ping packets reflects the changes caused by the transmitter. The receiver periodically sends ping packets to the ping reflector, tracking the latency of the network. The details about how the receiver uses the latency measurements to decode the data are explained in Section 3.4.

The receiver has different characteristics compared to typical RF receivers that provide a unique challenge in the context of monitoring latency. In the case of our receiver, the signal-to-noise ratio (SNR) is different from a traditional RF receiver. In our system, the noise is the natural latency of the network. The higher the natural latency of the network, the higher the noise of our system. The signal is the ability of our system to create a spike of latency above the natural latency (i.e., noise floor). The ratio of the latency spike to the natural latency of the network is our SNR.

3.2.1 Pinging Method. We consider two options for measuring the latency of the network. The first is using the `ping` utility. This method sends an ICMP echo request to the ping reflector and the ping reflector responds back with an ICMP echo response. The receiver measures the round trip time to measure the latency of the network. Because of the popularity of the `ping` utility and its potential for use in attacks [26], some devices have blocked ICMP traffic or network administrators might rate limit these packets [34].

To avoid this problem, we use TCP SYN packets instead. This method is the default scanning approach to the Nmap tool [30]. The ping reflector will either respond with a SYN/ACK if the port is open or RST if the port is closed. We measure the time from when we send the TCP SYN packet to when we hear a response. This method side-steps the problem with ICMP traffic and allows us to ping at any rate we would like.

3.2.2 Effective Pinging. Unlike traditional receiver systems, sampling the channel is not a benign operation, which makes our system challenging. A typical receiver's sample rate is limited by its hardware capabilities. However, in our system, the sampling rate is limited to the network capacity, and sampling the channel too often can cause adverse effects on the network. Since each "sample" causes two packets to be sent on the network (ping request and ping response), sampling too often will cause congestion, thus increasing the latency of the network. If the receiver samples too aggressively, it will jam itself. The limiting factor for our system is not how fast we can sample the network (send pings), but how often we can sample the network *without causing a negative effect on the network*. We discuss the rate at which we send pings in Section 5.

3.2.3 Parallel Pings. Traditional RF receiver systems typically have a constant sample rate, however, our system does not have that benefit. If we send a ping and wait for its response (i.e., sample the channel), then our sample rate will change depending on the latency of the network. When the latency is low, our sample rate will be faster and when the latency is high, our sample rate will be slower. To alleviate this problem, we send and receive pings in parallel. By not blocking each ping on the reception of the previous one, we can send pings at a constant rate. Each received ping response is aligned with its corresponding sent ping to provide a high-fidelity representation of latency over time. To do this, we overload the use of the TCP sequence number as a way of providing a unique ID. The insight for this to work is that the responding TCP device responds back with an ACK number that is one more than the sequence number, allowing us to overload its meaning and keep its value between the request and response without adding any logic to the ping reflector. When a TCP SYN packet is sent, a random sequence number is inserted into the packet. The response TCP SYN/ACK or RST packet contains an ACK number with the random number, incremented by one. The receiver matches the response with the corresponding request ID and measures the round trip time. By doing so, we can provide a more consistent sample rate to the receiver.

3.3 Ping Reflector

The purpose of the ping reflector is to have a wireless device that allows the receiver to measure the latency of the wireless network. The ping reflector can be any wireless device that is already part of the network. The ping reflector is unaware of LSK and its only purpose is to respond to ping requests. Using a third device is an approach used by others, such as Amazon, to create an application called "Wifi Simple Setup" [8]. We show in Section 6 that the jamming on the ping reflector has very little impact on its overall throughput.

3.4 Data Modulation

Encoding data in the latency of a network presents exciting challenges due to the interdependence between the various modulation parameters. Figure 5 illustrates a single controlled spike in latency that is created by injecting a pair of NULL frames. The top graph shows the jam from the latency domain while the bottom graph shows the same jam from the number of received packets domain. The latency increases from the baseline l_0 to l_1 at time t_1, which corresponds to the first NULL frame. Latency decreases back to the baseline over the duration of the jamming interval T_j (top graph

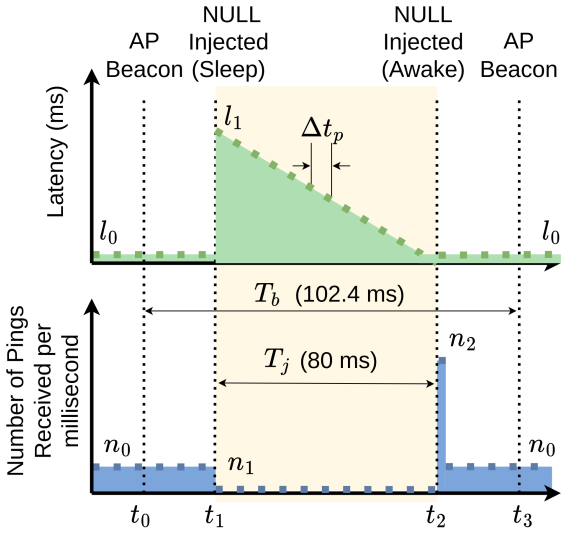

Figure 5: The effect of NULL frame jamming on the network latency and the number of RX pings per millisecond, over the course of a single beacon interval.

of Figure 5). This triangle shape makes it harder to detect using conventional signal processing techniques. To solve this problem, we instead use the number of pings received per millisecond (bottom graph of Figure 5). This creates a sharp spike after the jamming occurs. When the network is un-jammed at t_2, all buffered pings are received in less than 1 millisecond, and we can observe the spike in received traffic due to the jamming as a single data point n_2, occurring at time t_2. This method provides a more time-deterministic representation of the modulated latency, and is used by the receiver and decoder, as described in Section 5.

3.4.1 Signal to Noise Ratio (SNR). We observe that the signal to noise ratio (SNR) of our system in ideal conditions would be represented as the ratio of n_2/n_0. Since n_2 represents the number of buffered packets over T_j, all received within 1 ms, we can maximize the SNR by increasing T_j. However, as discussed in Section 3.1.4, if T_j exceeds the beacon interval, the ping reflector will automatically become un-jammed at t_3. We maximize the SNR in a single beacon interval by selecting T_j to be as large as possible without interfering with the next beacon.

3.4.2 Encoding. If one bit were sent per beacon interval, and every attempt to jam the channel is successful, then we would achieve a data rate of $\frac{1 \text{ bit}}{102.4 \text{ ms}} = 9.76$ bps. However, there are network phenomena that interfere with this encoding scheme and cause an unpredictable degradation of SNR. Data can be buffered naturally by either the router, ping reflector, or the networking stack on the receiver, resulting in latency spikes not triggered by an LSK transmitter. The NULL frames are injected into the network without any guarantee of being received by the access point, and are occasionally missed, resulting in a missing latency spike for a given beacon interval.

To increase the overall SNR of our system, reduce bit error, and deal with naturally occurring latency spikes in the network, we encode data using a pseudo-noise (PN) code similar to direct sequence spread spectrum. The transmitter and receiver share the code that is used to encode/decode the data. This provides coding gain to our system allowing us to decode data in spite of a capped SNR and external network interference. In this context, the transmitter encodes the data by multiplying a stream of data with the PN code. To send a binary one, the transmitter sends the PN code and to transmit a binary zero, the transmitter sends the inverse of the PN code. The PN code consists of chips, with a one chip being the presence of high latency and a zero chip being a normal value of latency.

We use two different PN codes, one for syncing the transmission with the receiver and one for spreading the data. We select a 31-bit maximal length sequence (MLS) code to use as the sync word in order to give a strong autocorrelation [31] while searching for the packet. To encode each of the data bits to be transmitted, 11-bit Barker Codes are used to provide a balance between an adequate autocorrelation [21] and preserving data rate.

Figure 6 shows a flow of data through the entire system. The data, spreading code, and sync word shown in the figure are small examples to illustrate the flow of data and are not the actual values used in our system. The transmitter takes arbitrary data and multiplies it by the PN code, which we call our spread data. The transmitter prepends a sync word to the front of the packet. The transmitter then goes through each of the chips in the sync word and spread data. If the chip is a one, we jam the network. If the chip is a zero, we do nothing and wait for a chip interval for the next chip. To jam the network, we wait for a beacon to be transmitted and then send a NULL frame to jam the ping reflector. During this time, the receiver is pinging the ping reflector, measuring the pings per millisecond. It correlates the samples with the sync word. When the sync word is detected, it switches to the spreading code to decode the data. Using this method, the transmitter is able to encode data into the latency of the network and the receiver is able to decode the data.

3.4.3 Data Rate. By transmitting a single chip per beacon interval, and using an 11-bit Barker code that generates 11 chips per bit of data to transmit, we are able to transmit a single bit per 11 beacon intervals. This results in an overall system data rate of $\frac{1 \text{ bit}}{0.1024 \cdot 11} = 0.89$ bps. This is slow compared

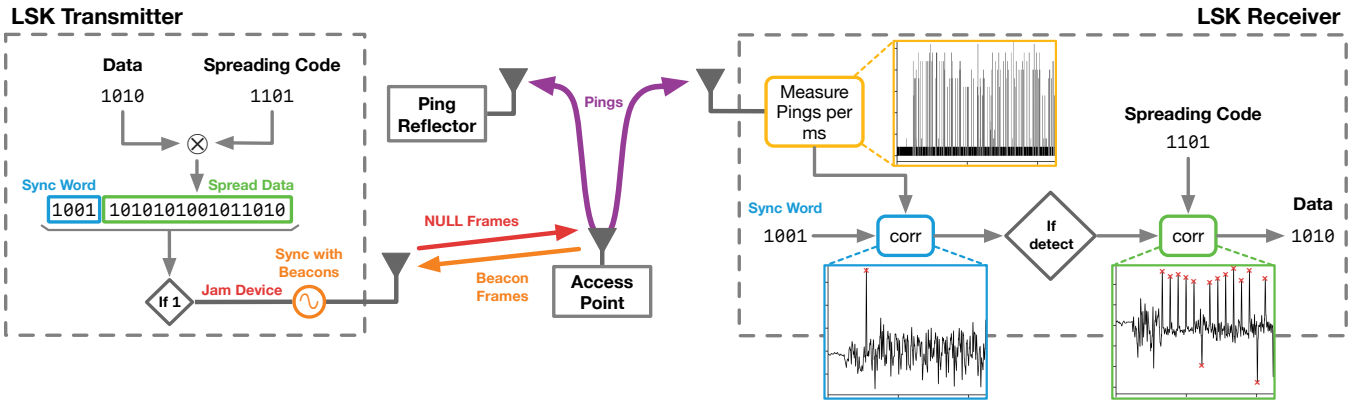

Figure 6: Flow of data between different nodes of LSK.

to typical WiFi data rates, however, for the context of WiFi sensors, this data rate works great. For example, a sensor that reports data every 5 minutes would have more than enough bandwidth to send its data. For these types of devices and use cases, having a slow data rate is not a problem.

4 WICKET: NETWORK PARTITIONING USING LSK

LSK is a general-purpose modulation scheme that encodes data in the latency of the network. We next describe Wicket, our application layer protocol that uses LSK to partition the network for IoT devices. Since WiFi does not easily provide fine-grained access control to devices, Wicket fills in the gap. It allows for an untrusted WiFi device to send data to a willing receiver on the trusted network.

The user starts the Wicket application on the receiver, signaling that they want to start receiving data from a Wicket transmitter. The receiver must first select a suitable ping reflector. A good ping reflector is a device that is wireless but has low variation in latency. The IP address of the ping reflector can be provided by the user of the application, determined through some other means (such as what is done with Amazon's "Wifi Simple Setup" [8]), or discovered automatically by the receiver. Once the receiver has selected a ping reflector, it starts sending ping messages, as described in Section 3.2.1, to sample the network's latency. The Wicket application is also set up to forward the data it receives to another node or display it.

4.1 Threat Model

The purpose of Wicket it to side step a potential security threat of providing a network's credentials to an untrusted device, giving it full access to the network. In this section we describe two threat scenarios. One where an adversarial device connects to a network using Wicket and another where an adversary attacks the communication between an LSK transmitter and receiver.

By serving as a low-bandwidth channel between a sensor and an application on the network that is expecting only a certain data type, the ability of the new device to compromise the WiFi network security is significantly limited. The device is partitioned from the rest of the network by the Wicket application that is receiving the data, so it can not contact other devices directly. The Wicket application is designed to only receive data from the sensor and upload it to a server. This puts the user in control of where the data is sent by the Wicket application. An attacker would need to compromise the Wicket application, changing its behavior, in order to compromise the network, which is much more challenging compared to compromising a typical WiFi network.

In terms of an adversary attacking the communication between an LSK transmitter and receiver, the main areas of concern for the security of our protocol would be maintaining the confidentiality, integrity, and availability of the sensor data itself. If needed, the data from the sensor could be encrypted and integrity protected to guarantee confidentiality and data integrity. An attacker could compromise the availability of the LSK data by causing network interference, but the same threat vector applies to normal WiFi communication. LSK provides no additional security vulnerabilities compared to typical wireless communication and standard security practices can be used to protect the data.

4.2 Bootstrapping

The transmitter has a bootstrapping problem. How does it know the MAC address of the ping reflector in order to try to jam it? To solve this problem, when the receiver sends a ping, it inserts a unique identifying source MAC address into the Ethernet frame, instead of its own MAC address. The transmitter goes into monitor mode which lets it receive

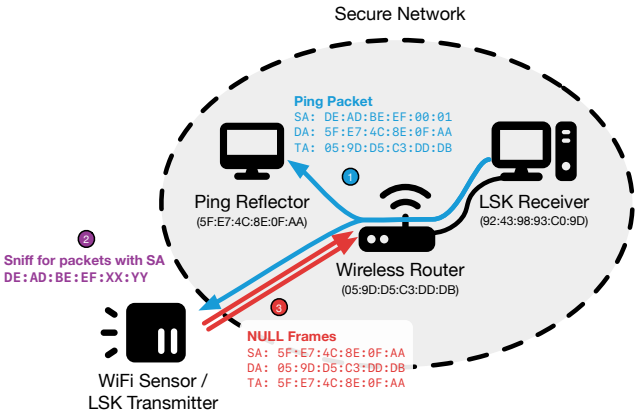

Figure 7: The bootstrapping procedure for Wicket. The ping packet contains all the information the transmitter needs to start jamming the ping reflector.

raw 802.11 frames, looking for these unique source MAC addresses. Two key insights are required to make this possible. First, WiFi is inherently broadcast, so even if a ping packet is unicast, it will be broadcast wirelessly for any device listening to receive. Second, though the transmitter, who is untrusted and not part of the secure network, can not decode the payload of packets since they are encrypted, *it can see the source and destination MAC addresses of packets, which are not encrypted.* WiFi does not encrypt the link layer header information. Detecting one of these frames indicates that to the transmitter there is an LSK receiver present on the channel. It also gives the transmitter all of the information it needs to know the ping reflector's MAC address. In spite of the receiver using an incorrect source MAC address, the ping packets still get returned to the receiver because the correct IP address is used.

The flow of information is shown in Figure 7. The receiver sends a ping packet with the correct IP addresses and correct destination MAC address, but an identifying source MAC address. The AP inserts its MAC address into the transmitter address of the 802.11 frame [2] and sends the packet wirelessly to the ping reflector. The ping reflector responds to the ping request and sends its response to the receiver. During the same time, the transmitter enters monitor mode, sniffing for frames with a specific source MAC address. Once it detects such a frame, it now knows the ping reflector's MAC address (the source address of the frame) and the network's AP's MAC address (the transmitter address of the 802.11 frame). This is all of the information the transmitter needs to know to start encoding data into the latency of the network.

4.3 Multiple Access Control

As mentioned previously (Section 3.1.3), sending NULL frames to different devices acts as an orthogonal channel. We can use this ability to provide multiple access control in two ways, channel partitioning (using multiple ping targets) and time division (using the same ping target). If multiple sensors want to send data at the same time, then they can target different ping reflectors with the Wicket receiver pinging two devices in parallel. To provide channel partitioning and time division capabilities, we develop a "packet structure" for conveying important information to the transmitter. We use the form, `DE:AD:BE:EF:XX:YY`, where `DE:AD:BE:EF` is an identifier for Wicket, `XX` represents any unique identifier and `YY` represents if the channel is already being used or not. Together with the source address to identify the ping reflector and the transmitter address to identify the AP, the transmitter is able to gain detailed information about the state of Wicket on the network. `XX` can be used to uniquely identify a specific deployment or location so that if multiple Wicket networks are deployed nearby, they can differentiate themselves. This is programmed by the user through the Wicket receiver. The receiver changes `YY` from `00` to `01` when a sync word is detected and the channel becomes busy. The transmitter looks for a source MAC address with that form and can identify how many channels are available and which channels are currently being used. If all the channels are full, multiple transmitters can also be multiplexed in time using the `YY` field as a way to know if the channel is busy. For example, once a channel moves from busy to empty, the transmitter can wait a random back-off and if the channel is still not being used, can start sending its data.

Depending on the needs of the transmitters, a single ping reflector can handle multiple devices. If the channel is too busy, additional ping targets can be added to provide additional channels. Using the destination MAC address as a way to pass information to the transmitter provides much flexibility for Wicket to adapt to the needs of the WiFi sensors and network.

5 IMPLEMENTATION

5.1 Transmitter

We select the Espressif ESP32 development board as our Wicket transmitter because it is a popular WiFi SoC for IoT devices, provides a bare metal system to get more accurate timing characteristics, and provides a direct API for the WiFi module. Since the transmitter needs to send the NULL frames timed relative to the beacon, with response times less than 10 ms, a real-time system must be used. In a traditional operating system, the delay from when a beacon is received by the WiFi chipset and it is processed by the application is non-deterministic and often larger than the 10 ms window

that we need. By utilizing the ESP32, our transmitter application can run bare metal and provide the precise timing required to induce buffering on the network in a consistent manner. The data to be transmitted is first spread into an array containing the 1's and 0's for each chip in the 31-bit MLS sync word, followed by chips of 11-bit Barker codes for each of the application data bits to be transmitted.

Prior to transmission, the ESP32 sniffs all visible packets in the air, searching for the Wicket code word in the source MAC address, as described in Section 4. When the code word is detected in a packet, it is assumed that the packet is a valid TCP packet being sent to the ping reflector to be spoofed. The transmitter then uses the destination MAC address from that packet as the source address in a new NULL frame packet. The destination MAC address for this new packet is the MAC address of the AP, which is also obtained from the sniffed packet.

The transmitter then begins a state machine that is triggered by the reception of each beacon frame. A timer is also used to trigger the state machine in the event that one or more beacons are missed. This results in very precise timing that allows us to synchronize the injection of the NULL frames and have precise control over the introduced latency. The state machine is initialized with a pointer to the first bit in the encoded data to transmit. For each iteration, the following actions are taken: 1) If the current bit to transmit is a '1', transmit a NULL frame triggering the AP to start buffering frames for the ping reflector. Delay for T_j ms, then send another NULL frame to release the buffered frames. Increment the data pointer. 2) If the current bit to transmit is a '0', then do nothing, increment the data pointer, and wait for the next state machine iteration. The jamming interval T_j is selected to be 80 ms, as that provides an adequate SNR while not risking interference from the next beacon.

5.2 Receiver

The receiver is implemented in Python using a Linux-based laptop computer, connected via Ethernet to the WiFi router. A ping interval Δt_p of 5 ms between pings is selected for our implementation, which strikes a balance between high-fidelity latency sampling and SNR while not adding too much load to the network.

The receiver measures the network latency by using the Python multiprocessing and scapy [11] libraries in the following way. First, two processes are started. One of the processes is used to monitor all traffic on the Ethernet interface. The second process sends TCP SYN packets to all the ping reflectors, with a 5 ms delay between packets. As responses come back to the receiver, the first process matches each TCP SYN packet with the corresponding TCP RST packet (or SYN/ACK packet if the port is open) based on the sequence number, calculating the round-trip-time (RTT). This data is then sent to the decoder for processing.

5.3 Decoder

The decoder begins by constructing a new 2-dimensional array with the first dimension consisting of 1 ms intervals that span the duration of the capture. Then, for each interval, the number of response TCP packets received within the interval is recorded in the second dimension. To make correlation easier, the data set is then further conditioned by taking a rolling variance across the data. Using variance provides a stronger impulse response when the latency increases due to our jamming.

There are two correlation operations that need to happen to decode the LSK message: sync word and message payload. Both the sync word and the message payload codes are individually correlated with the smoothed variance data. After correlation is performed for both codes the data for each is zero meaned. The decoder synchronizes if the sync word correlation data exceeds a threshold of two times the standard deviation of the data. If synchronization occurs, the data can be demodulated from the Barker code correlation data.

When the payload data is modulated it is synchronized with the beacon intervals of the AP. This eases the demodulation of the data since we do not worry about our ping interval and transmitter jamming getting out of sync. From the point of synchronization of the sync word, the Barker code correlation data is sampled every 102 samples since the data is broken into 1 ms windows and beacon frames come every 102.4ms [2]. At each sample point, the demodulation algorithm adaptively adjusts where it samples based on nearby peaks in correlation to adjust for sampling jitter. Data bits are then determined based on whether the correlation value is above or below zero, one being above and zero below.

6 EVALUATION

We evaluate our protocol in several ways to understand its flexibility and reliability. For most evaluation, we evaluate Wicket since it tests the whole system which uses LSK. By evaluating Wicket we are also evaluating LSK.

6.1 Evaluation Setup

To verify the performance of Wicket and LSK, data is sent from the transmitter to the receiver, and the received data is compared against the expected data to generate a bit error rate (BER). To demonstrate that our system is not dependent on input data, 10 randomly generated 32-bit sequences are used as payloads to transmit. The size of the data payloads

was selected to represent a single packet containing compressed sensor data. A test consists of each of the 10 sets of data being sent from the transmitter to the receiver, and the calculated BER for each data set is averaged together to provide a single result for the test.

Evaluation was performed in a home WiFi environment with interference levels typical of a residential IoT sensor application. WiFi channel 11 is selected for testing. As discussed in Section 6.3, The baseline channel utilization rate as reported by the access point is approximately 20%.

For the tests used in the evaluation, channel utilization is measured by monitoring the QBSS Load Information Element (IE) of the beacon frame, which reports the channel utilization rate as an 8-bit value. This was averaged together over 1-second intervals, and collected for the duration of each test.

6.2 Interoperability With Various WiFi Routers

We design Wicket and LSK to not require any custom firmware or manufacturer-specific features of the WiFi router, relying on core features of the 802.11 protocol. As such, we expect that Wicket should perform well on a wide variety of WiFi routers. To test this, we verify that our protocol can be run using three different routers, as shown in Table 1. We select these routers because they represent a broad spectrum of router capabilities, brands, and costs. In all cases, our protocol consistently achieves a bit error rate of zero on all three routers under typical conditions. This shows that our protocol utilizes fundamental WiFi features and is not dependent on router or protocol behavior.

6.3 Effect of Existing WiFi Network Traffic on Wicket Performance

As discussed in Section 3.4, buffering in an LSK network occurs not only as the result of the NULL frames injected by the transmitter but by naturally occurring network conditions as well. In WiFi environments with low network load, nearly all buffering is caused by the transmitter. However, when the network is more congested, buffering may occur naturally on either the AP or the ping reflector. In both cases, this can interfere with the timing of the LSK-induced buffering and introduce bit errors into our communication. This section evaluates at what point external network congestion will degrade the LSK communication channel.

To measure this, the WiFi network is loaded with traffic in a controlled manner using iPerf [15]. While iPerf is typically used to measure the maximum throughput of a network, it also allows an input argument to specify a target throughput to test and will maintain that throughput consistently through the duration of the test. An iPerf server was launched on a test computer connected to the WiFi router via Ethernet, and an iPerf client was launched on a device connected to the 2.4 GHz WiFi network. This topology allows us to place a controlled load on the 2.4 GHz network simulating external network traffic.

Router Model	WiFi Version	Cost
Linksys MR7500	WiFi 6E	$230
Netgear Nighthawk AC2600	WiFi 5	$154
TP-Link Archer AC1200	WiFi 5	$40

Table 1: A table of routers used for testing, with their WiFi version and cost.

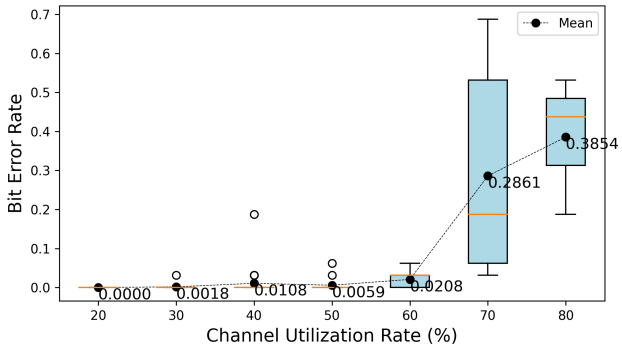

Figure 8: Effect of network congestion on the bit error rate (BER) of LSK communication

The channel is first characterized by running iPerf at maximum speed, which for this evaluation indicates a maximum channel capacity of 80 Mbps. A series of tests are then performed while the iPerf client streams data at 10 Mbps, 20 Mbps, 30 Mbps, and so forth up to the point where the Wicket communication is observed to break down. The baseline channel utilization rate with no Wicket communication and no iPerf load is 20% as reported by the AP. Because the observed channel utilization is not constant but varies by up to 15% over time, the test results are grouped by the observed utilization rate from when each data set is recorded.

Figure 8 shows the distribution of test results of Wicket for each of the observed utilization rates. It can be seen that for the topology and encoding used, the data was successfully received up to a 60% utilization rate. Since WiFi networks are typically designed such that the target airtime utilization remains below 50-60% [13, 33], these results demonstrate the ability of Wicket to perform well in a wide variety of network loading conditions. Except for the most extreme networks, Wicket will perform with very low bit error.

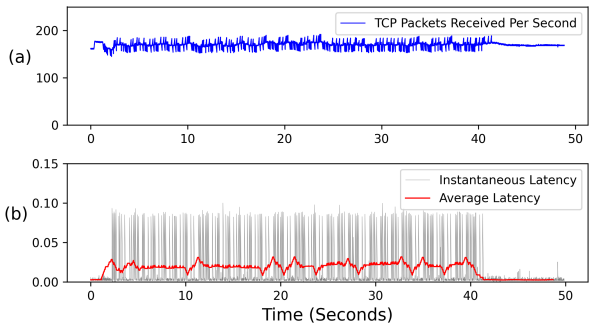

Figure 9: Measured change in (a) average network latency and (b) TCP packets received per second as a result of a single LSK transmission

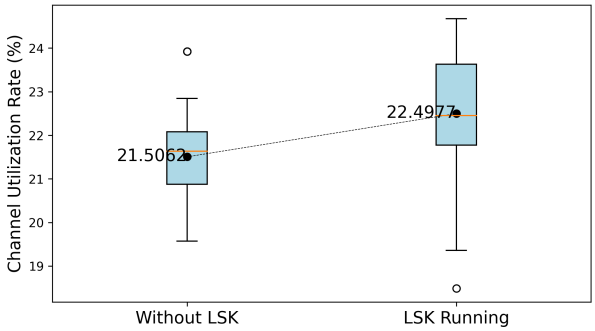

Figure 10: The effect of running Wicket/LSK compared to a idle network.

6.4 Effect of Wicket on Network Performance

Because Wicket is intended to be used on existing WiFi networks, it is important that the protocol does not place a disproportionate load on the network or otherwise slow it down for other devices. We evaluate the effect of Wicket on the device being pinged and the network utilization rate.

6.4.1 Effect on Ping Reflector Network Performance. Figure 9 shows the average latency (bottom graph) and the total TCP packets received per second (top graph) over the course of a single Wicket transmission. While the jamming by the Wicket transmitter increases the average latency by about 25 milliseconds, it is noted that the number of TCP packets received per second remains the same during transmission and afterward. This highlights the fact that LSK causes only minor increase in latency, and the overall network performance of the ping reflector is not impacted in a way that would significantly degrade most applications.

6.4.2 Effect on Utilization Rate. To evaluate the effect of the Wicket protocol on the utilization rate of a WiFi network as a whole, we compared the channel utilization rate measured on an idle network to that measured while a single device was transmitting. In both cases, the utilization rate reported by the AP was recorded over 45 seconds.

The test results are shown in Figure 10, and indicate that for a single transmitter/receiver running Wicket, the marginal network load is about 1%. This is not surprising, since the TCP packets are less than 60 bytes. Considering both the TCP SYN and RST/ACK packets, which are sent over the 2.4 GHz WiFi network every 5 milliseconds, this only constitutes 0.192 Mbps of load placed on the network. While our utilization is potentially higher than most WiFi sensors, the ability to protect a network from an untrustworthy sensor can be worth the cost of slightly extra utilization.

6.5 Wicket Performance at Various Ranges

Because Wicket is intended to be used in wireless applications, in this section we evaluate the effect of range between the transmitter and access point (AP). For this evaluation, we assume that the ping reflector device has been selected to have a solid and reliable connection to the access point.

Since in wireless communication, multi-path and line-of-sight obstructions can interfere with physical range measurements, received signal strength as seen by the transmitter is used. The ESP32 transmitter is connected to a Raspberry Pi test device. The ESP32, upon receiving beacons from the access point, measures the RSSI and reports it to the Raspsberry Pi, which sends it as an MQTT message to the test computer. For each test iteration, the transmitter is moved farther away from the AP, the RSSI as seen by the transmitter is logged, and the results are grouped by RSSI.

The results are given in Figure 11, and indicate strong Wicket performance until an RSSI of about -60 dBm. It should be noted that upon inspection of the raw data, this appears to be about the point where the AP stops seeing the NULL frames sent by the ESP32. Given that the ESP32 continued to receive the AP beacons and report RSSI, it may be concluded that the drop-off in performance is likely caused by a limitation in transmit power on the transmitter.

6.6 Scaling to Multiple Wicket Devices

To show that Wicket is viable for multiple sensors on one network, the test setup is expanded to support multiple transmitters running in parallel, communicating with a single receiver as described in Section 4. Figure 12 shows the average bit error rate (BER) of tests run with 1, 2, 3, and 4 devices transmitting simultaneously. We stop at four concurrent transmitters because of the lack of equipment to test more than four nodes. The results demonstrate consistently low bit error rates even as additional concurrent transmissions are added. This is to be expected from the finding in

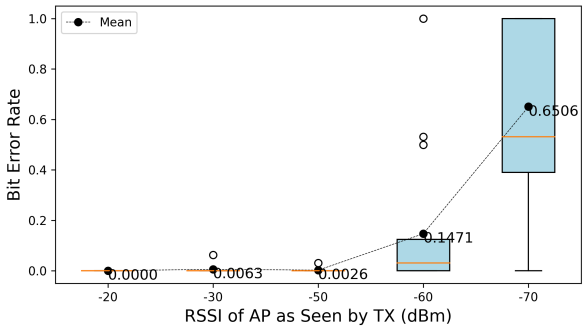

Figure 11: The effect of range between the LSK transmitter node and the AP on the bit error rate (BER).

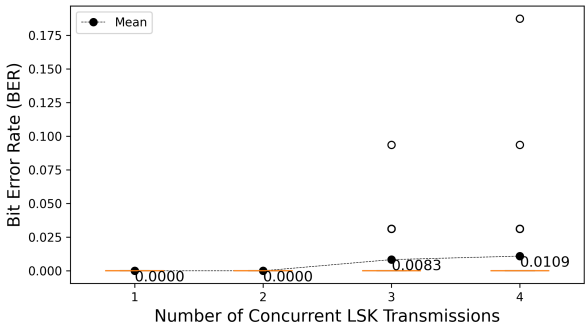

Figure 12: The effect of increasing the number of parallel LSK devices on the bit error rate (BER).

Section 6.4 that each transmission adds less than 1% to the network utilization rate. This evaluation validates the use of multiple transmitters and indicates that devices can be added to a Wicket network at minimal cost.

7 RELATED WORK

As mentioned in Section 2, advanced WiFi systems can be set up to partition networks between a trusted network and an untrusted network. However, this requires advanced networking knowledge and equipment.

Much work, both commercially and academically, has been done to improve the ease of connecting a device to a WiFi network. Such efforts include WPS [1], Espressif's Wi-Fi Easy Connect [5], and Secure Transfer of Association Protocol [28]. These technologies work by creating a secondary channel to provision a device unto a network. However, that does not solve the underlying problem of trust. Once the device is connected to the network, it has access to the whole network. Our solution is solving a problem of security/trust.

Others have created side channels for unassociated WiFi devices to communicate. For example, some IEEE 802.11 frames, such as a beacon, or 802.11 fields, such as the source MAC address, can be used to insert arbitrary data (typically called bit stuffing) [19]. However, such an approach requires both the sender and the receiver to be wireless devices. LSK does not require the receiver to be wireless—it works over a wired connection as well. By allowing the receiver to be wireless, our system is unable to use IEEE 802.11 specific frames or fields. Prongle [35] can be used to set up communication between two unassociated wireless devices but it requires a companion device. LSK does not require extra hardware and is focused on setting up communication between an unassociated device and a *secure network*. Also, LSK does not require the receiving node to be wireless. The receiving node can be wireless or wired, meaning LSK can traverse multiple link layer protocols. WiPush [6] provides a system for sending push notifications from a network to an unassociated device. LSK addresses the opposite problem: how to send data *into* a secure network. ONPC [29] used beacon frames and noise on the channel to set up communication. However, the focus of this work was on extending the range of communication and requires the receiver to be able to measure the noise on the WiFi channel. LSK only requires the receiver to be able to measure the latency of the network.

Our work has similarities to cross-technology communication protocols, such as FreeBee [24], C-Morse [38], and E-Sense [12]. These methods modulate data in packet timings or packet lengths, rather than latency. To the best of our knowledge, we are the first to encode data in the latency timing of a wireless network as observed by a wired device.

8 CONCLUSION

With WiFi sensors becoming lower cost, it is easy to integrate them into your network without giving much thought to the potential security risk that these devices present. A rogue device can infiltrate a network looking for weak nodes or act as a DDoS bot. WiFi makes it hard for a device to send data on a network without giving that device complete access to your network. LSK is a novel way of encoding data into the latency of a wireless network, allowing an outsider device to send data into a trusted network. We use intricate knowledge about the 802.11 protocol and frame types to surgically jam one device, leaving the rest of the network unaffected. We develop Wicket that demonstrates the use of LSK by creating a new security association between an untrusted WiFi device and a secure network. All innovations outlined in this work require no changes to a device's hardware and provide easy integration into currently deployed systems.

ACKNOWLEDGEMENTS

We thank the anonymous reviewers and shepherd for their valuable comments and suggestions. This work is funded by NSF award 2153317.

REFERENCES

[1] 2013. https://www.cisco.com/assets/sol/sb/WAP561_Emulators/WAP561_Emulator_v1.0.4.1/help/Wireless12.html

[2] 2021. IEEE Standard for Information Technology–Telecommunications and Information Exchange between Systems - Local and Metropolitan Area Networks–Specific Requirements - Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications. *IEEE Std 802.11-2020 (Revision of IEEE Std 802.11-2016)* (2021), 1–4379. https://doi.org/10.1109/IEEESTD.2021.9363693

[3] 2022. https://en.wikipedia.org/wiki/Wicket_gate

[4] 2022. Linux Wireless - Monitor Mode. Retrieved August 24, 2023 from https://wireless.wiki.kernel.org

[5] 2023. https://docs.espressif.com/projects/esp-idf/en/latest/esp32/api-reference/network/esp_dpp.html

[6] Utku Günay Acer and Otto Waltari. 2017. WiPush: Opportunistic Notifications over WiFi without Association. In *Proceedings of the 14th EAI International Conference on Mobile and Ubiquitous Systems: Computing, Networking and Services (MobiQuitous 2017)*. Association for Computing Machinery, New York, NY, USA, 353–362. https://doi.org/10.1145/3144457.3144492

[7] Salah A. Alabady, Fadi Al-Turjman, and Sadia Din. 2020. A Novel Security Model for Cooperative Virtual Networks in the IoT Era. *International Journal of Parallel Programming* 48, 2 (April 2020), 280–295. https://doi.org/10.1007/s10766-018-0580-z

[8] Amazon. 2023. Amazon Frustration-Free Setup Frequently Asked Questions. https://www.amazon.com/gp/help/customer/display.html?nodeId=GMPKVYDBR223TRPY

[9] Ioannis Antzoulis, Md Minhaz Chowdhury, and Shadman Latiff. 2022. IoT Security for Smart Home: Issues and Solutions. In *2022 IEEE International Conference on Electro Information Technology (eIT)*. 1–7. https://doi.org/10.1109/eIT53891.2022.9813914

[10] Ioannis Antzoulis, Md Minhaz Chowdhury, and Shadman Latiff. 2022. IoT Security for Smart Home: Issues and Solutions. In *2022 IEEE International Conference on Electro Information Technology (eIT)*. 1–7. https://doi.org/10.1109/eIT53891.2022.9813914

[11] Philippe Biondi. 2023. Welcome to Scapy's documentation! https://scapy.readthedocs.io/en/latest/

[12] Kameswari Chebrolu and Ashutosh Dhekne. 2009. Esense. *Proceedings of the 15th annual international conference on Mobile computing and networking* (2009). https://doi.org/10.1145/1614320.1614330

[13] Cisco. 2010. Voice Over Wireless LAN (VoWLAN) Troubleshooting Checklist. https://www.cisco.com/c/en/us/td/docs/wireless/technology/vowlan/troubleshooting/VoWLAN_Troubleshooting_Checklist.html

[14] Cisco. 2022. Cisco annual internet Report - Cisco Annual Internet Report (2018–2023) White Paper. https://www.cisco.com/c/en/us/solutions/collateral/executive-perspectives/annual-internet-report/white-paper-c11-741490.html

[15] Jon Dugan, Seth Elliott, Bruce A. Mah, Jeff Poskanzer, and Kaustubh Prabhu. 2023. iPerf - The ultimate speed test tool for TCP, UDP and SCTP. https://iperf.fr

[16] Kevin Eustice, Leonard Kleinrock, Shane Markstrum, Gerald J. Popek, Venkatraman Ramakrishna, and Peter L. Reiher. 2003. WiFi Nomads and their Unprotected Devices : The Case for QED — Quarantine , Examination , and Decontamination. https://api.semanticscholar.org/CorpusID:16546812

[17] Matthew S. Gast. 2005. *802.11 Wireless Networks: The Definitive Guide* (2nd ed.). O'Reilly Media, Inc.

[18] Wenjun Gu, Zhimin Yang, Dong Xuan, Weijia Jia, and Can Que. 2010. Null Data Frame: A Double-Edged Sword in IEEE 802.11 WLANs. *IEEE Transactions on Parallel and Distributed Systems* 21, 7 (2010), 897–910. https://doi.org/10.1109/TPDS.2009.96

[19] Vishal Gupta and Mukesh Kumar Rohil. 2013. Bit-stuffing in 802. 11 beacon frame: Embedding non-standard custom information. *International Journal of Computer Applications* 63, 2 (2013).

[20] Stephen Herwig, Katura Harvey, George Hughey, Richard Roberts, and Dave Levin. [n.d.]. Measurement and Analysis of Hajime, a Peer-to-peer IoT Botnet. *Network and Distributed Systems Security (NDSS) Symposium* ([n. d.]). https://doi.org/10.14722/ndss.2019.23488

[21] Md. Alamgir Hossain and S. M. A. Islam. 2012. PERFORMANCE ANALYSIS OF BARKER CODE BASED ON THEIR CORRELATION PROPERTY IN MULTIUSER ENVIRONMENT. https://api.semanticscholar.org/CorpusID:61268232

[22] Jacob Johnson. 2024. Wireless Latency Shift Keying Repository. https://github.com/NET-BYU/wireless-latency-shift-keying

[23] Bryce Kellogg, Aaron Parks, Shyamnath Gollakota, Joshua R. Smith, and David Wetherall. 2014. Wi-Fi Backscatter: Internet Connectivity for RF-Powered Devices. In *Proceedings of the 2014 ACM Conference on SIGCOMM (SIGCOMM '14)*. Association for Computing Machinery, New York, NY, USA, 607–618. https://doi.org/10.1145/2619239.2626319

[24] Song Min Kim and Tian He. 2015. Freebee. *Proceedings of the 21st Annual International Conference on Mobile Computing and Networking* (2015). https://doi.org/10.1145/2789168.2790098

[25] Constantinos Kolias, Georgios Kambourakis, Angelos Stavrou, and Jeffrey Voas. 2017. DDoS in the IoT: Mirai and other botnets. *Computer* 50, 7 (2017), 80–84.

[26] Sanjeev Kumar. 2007. Smurf-based Distributed Denial of Service (DDoS) Attack Amplification in Internet. In *Second International Conference on Internet Monitoring and Protection (ICIMP 2007)*. 25–25. https://doi.org/10.1109/ICIMP.2007.42

[27] Heju Li, Xin He, Xukai Chen, Yinyin Fang, and Qun Fang. 2019. Wi-Motion: A Robust Human Activity Recognition Using WiFi Signals. *IEEE Access* 7 (2019), 153287–153299. https://doi.org/10.1109/ACCESS.2019.2948102

[28] Philip Lundrigan, Sneha Kumar Kasera, and Neal Patwari. 2018. Strap: Secure transfer of association protocol. *2018 27th International Conference on Computer Communication and Networks (ICCCN)* (2018). https://doi.org/10.1109/icccn.2018.8487333

[29] Philip Lundrigan, Neal Patwari, and Sneha K. Kasera. 2019. On-Off Noise Power Communication. In *The 25th Annual International Conference on Mobile Computing and Networking (MobiCom '19)*. Association for Computing Machinery, New York, NY, USA, Article 35, 14 pages. https://doi.org/10.1145/3300061.3345436

[30] Gordon Lyon. 2023. Nmap. https://nmap.org

[31] D.V. Sarwate and M.B. Pursley. 1980. Crosscorrelation properties of pseudorandom and related sequences. *Proc. IEEE* 68, 5 (1980), 593–619.

[32] Sir Gawain. 2011. South gate of Friedestrom Castle with its wicket (pedestrian entrance). https://commons.wikimedia.org/w/index.php?curid=26781952

[33] Kaixin Sui, Siqi Sun, Yousef Azzabi, Xiaoping Zhang, Youjian Zhao, Jilong Wang, Zimu Li, and Dan Pei. 2016. Understanding the Impact of AP Density on WiFi Performance Through Real-World Deployment. In *2016 IEEE International Symposium on Local and Metropolitan Area Networks (LANMAN)*. 1–6. https://doi.org/10.1109/LANMAN.2016.7548845

[34] J. Udhayan and R. Anitha. 2009. Demystifying and Rate Limiting ICMP hosted DoS/DDoS Flooding Attacks with Attack Productivity Analysis. In *2009 IEEE International Advance Computing Conference*. 558–564. https://doi.org/10.1109/IADCC.2009.4809072

[35] Otto Waltari and Jussi Kangasharju. 2020. Prongle: lightweight communication over unassociated wi-fi. In *Proceedings of the 35th Annual ACM Symposium on Applied Computing (SAC '20)*. Association for Computing Machinery, New York, NY, USA, 994–1001. https:

//doi.org/10.1145/3341105.3375772

[36] Wei Wang, Alex X. Liu, Muhammad Shahzad, Kang Ling, and Sanglu Lu. 2015. Understanding and Modeling of WiFi Signal Based Human Activity Recognition. In *Proceedings of the 21st Annual International Conference on Mobile Computing and Networking (MobiCom '15)*. Association for Computing Machinery, New York, NY, USA, 65–76. https://doi.org/10.1145/2789168.2790093

[37] Huan Yan, Yong Zhang, Yujie Wang, and Kangle Xu. 2020. WiAct: A Passive WiFi-Based Human Activity Recognition System. *IEEE Sensors Journal* 20, 1 (2020), 296–305. https://doi.org/10.1109/JSEN.2019.2938245

[38] Zhimeng Yin, Wenchao Jiang, Song Min Kim, and Tian He. 2017. C-morse: Cross-technology communication with Transparent Morse coding. *IEEE INFOCOM 2017 - IEEE Conference on Computer Communications* (2017). https://doi.org/10.1109/infocom.2017.8057107