STRAP: Secure TRansfer of Association Protocol

Philip Lundrigan*, Sneha Kumar Kasera*, Neal Patwari†
*School of Computing
†Department of Electrical and Computer Engineering
University of Utah, Salt Lake City, Utah, 84112
*{philipbl, kasera}@cs.utah.edu, †npatwari@ece.utah.edu

***Abstract*—When several internet-of-things devices are required to be installed in a smart home, significant effort is required to provide each device with the association information for the home's wireless router. We design and build a novel protocol called Secure Transfer of Association Protocol (STRAP), which securely bootstraps connectivity between a set of deployed WiFi devices and a home's wireless router. We show that STRAP works in a variety of environments and is faster than conventional methods for connecting WiFi devices to home wireless routers.**

***Index Terms*—Internet of Things, Network security, Wireless networks**

I. Introduction

Low-cost internet-of-things (IoT) and "smart home" devices have applications in home security, home health care, aging-in-place, and energy efficiency. Several sensors and actuators may be required to be installed at one time in a home to enable one of these applications. For example, an aging-in-place system may use multiple motion sensors, chair or bed sensors, and smart pill dispenser. As another example, a smart HVAC control system may use motion sensors, temperature sensors, thermostat, CO_2 sensors, and air pollution sensors. In this paper, we are specifically motivated by an application in which sensors are installed in each of tens of thousands of homes for collecting air pollution exposure data in large human subject studies for epidemiological research into asthma and other diseases [1]. An installer may bring and connect these devices, or, alternatively, a kit may be mailed to the resident with instructions to be self-installed. It is critical for the installation to be fast and with as few steps as possible. An installer's time is expensive, and an untrained resident may be discouraged or confused by a long or complicated installation process.

Most IoT devices use the home WiFi to connect to the Internet. Typically, these devices do not have a keyboard or screen, making it much harder to provide it with information to access a home's existing encrypted WiFi network. Furthermore, many of these inexpensive devices, as is the case with our air quality sensors, have a WiFi network interface but no other network interfaces. Therefore, these devices cannot be accessed using any other network (e.g., Bluetooth) for installing the home's WiFi network name and password to allow them to access the home WiFi network.

One existing approach for dealing with our security "bootstrapping" problem is for the IoT device to act as a temporary access point and ask the user to connect to it and provide it the home WiFi network name and password. This is a common approach that many commercial devices use. While this approach works well when connecting one new device, the time to connect n devices scales linearly with n. Having to enter the network name and password into each device is cumbersome, time consuming, and error prone, especially when tens of devices are to be installed in each home. As IoT devices become more popular, we expect the number of devices that need to be installed to increase, making this problem worse. Another existing approach that uses enterprise WiFi security solutions (discussed in Section V) is typically not supported by consumer access points or home networks.

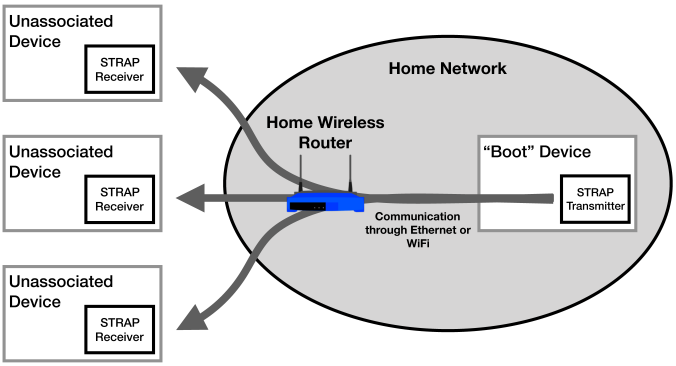

Fig. 1: Overview of STRAP components. The boot device securely sends the network name and password out of the home network to the unassociated devices.

In this paper, we address the above problem of fast and easy installation of IoT devices without using another network interface. Fundamentally, there is a "chicken and egg" problem here, as we need to securely transfer association information to the devices, but to be able to make that transfer through standard WiFi, each device needs to already have the association information and be connected to the network. We build a novel protocol that allows an already connected device (the "boot" device in Figure 1), either through Ethernet or WiFi, to securely transmit data to a group of unauthenticated devices. We use this ability to transmit the WiFi association information, allowing the unconnected devices to authenticate and connect to the network. We use key insights about how Ethernet and 802.11 frames are encapsulated, encrypted, and transmitted by wireless routers to encode the credentials into the Ethernet source and destination address fields. Our protocol provides

simplex communication to unassociated wireless devices by overloading the source and destination addresses of Ethernet and 802.11 frames, without requiring any modifications to the wireless router.

Figure 1 shows the high level components of our protocol. There are three components: a group of unassociated WiFi devices that are waiting to receive the authentication credentials, the home's wireless router, and an already connected device, the boot, which transmits the authentication credentials. The boot device must already be connected through Ethernet or WiFi so that it can send frames in the network. The shaded region represents the secured home network. Our approach works as follows: an installer enters the association information into the boot device. Since the device is already connected, it can send Ethernet frames into the network. The boot encrypts the association information and inserts this data into the source and destination addresses of multiple Ethernet frames, overloading the purpose of these fields. Unassociated, but listening devices, can receive this information, decrypt it, and connect to the wireless network.

Our protocol assumes there is a trust relationship between the boot and the devices that are trying to connect. Specifically, the boot has one set of pre-shared secret keys with all the unassociated devices. These keys are used for securing the communication between the boot and the unassociated devices. The pre-shared key has a similar role to the home's network name and password, but the benefit of using a pre-shared key is that the installer has complete control over the key. This allows the installer to pre-configure the devices with the key before they are set up in a home. In essence, we are using a second security association (the pre-shared keys) to bootstrap the WiFi security association. We describe the benefits of this design over other methods in Section V.

We envision the following two use cases for our work (though it is not limited to these use cases):

1) An installer comes to someone's home to install multiple WiFi devices. This could be part of a security system or an epidemiological study which involves deploying multiple environmental sensors in the home. All the devices that are part of the installation have been configured with a pre-shared key when flashed with software and prepared for deployment. The installer installs the devices where they need to go and turns them on. The installer then connects the boot device to the home's wireless router through Ethernet. The installer enters customer information, such as a customer ID, into the boot device which allows the boot device to securely download the secret key from the company's servers that is connected with the newly installed devices. The installer then asks the customer to enter in their WiFi network name and password. The boot device sends this information to all unassociated WiFi devices.
2) Similar to use case number one, but rather than an installer coming to the house, a box of WiFi devices are shipped to the customer for the customer to self-install. The customer places the devices and downloads an application on their smartphone or laptop. This device acts as a temporary boot device. The customer's device receives the shared key by either scanning a QR code that came with the devices or using login credentials and securely downloading the key from the company's servers. The customer then enters the WiFi network name and password into the temporary boot device, which broadcasts this information securely to all unassociated devices.

In this paper, we identify possible attack vectors to our protocol and show that it is resilient. In particular, we protect against an adversary that tries to trick devices into connecting to a rouge access point, as well as other threat vectors such as eavesdropping, packet manipulation, and replay attacks. Our protocol, based on only one-way transmission from the boot to the unassociated devices without any feedback, uses erasure coding to minimize the effects of wireless packet loss.

We evaluate our protocol in many different settings and setups. We show that our protocol works on a variety of wireless routers and has no effect on the network when run, and is much faster at setting up multiple devices compared to conventional approaches.

II. Secure TRansfer of Association Protocol

In order for a WiFi device to connect to a home's network, it must have the network name and password. Getting this information to devices without keyboards and screens is time-consuming, particularly when there are many devices to be connected. This problem will continue to grow worse as more IoT devices get deployed in homes as part of commercial systems and research studies. We especially found this to be a problem when we deployed air quality sensors in participant's homes as part of a pediatric asthma research study. As part of this study, we deployed ten air quality sensors in the home of each participating family. We found it took too long to deploy these sensors because a large part of the time involved connecting each sensor to the home's wireless network.

A common approach to solve this problem for commercial IoT devices is for a device to create its own temporary wireless network. This allows the person setting up the IoT device to connect to the temporary wireless network via smartphone. They select the network name they want the device to connect to, and enter the password, which then provides the IoT device the information it needs to connect to the WiFi network. Once the smartphone disconnects, the new device disables the temporary wireless network and connects to the home wireless router. This process works well with one new device, but it becomes a nuisance when installing multiple devices and increasingly untenable as the number of devices grows. Setting up each device one at a time requires a lengthy process for the installer and increases the chance that one device is set up incorrectly. A long installation is undesirable because of high labor costs, as well as the inconvenience for the resident. If devices are being self-installed by a customer, then a long and complicated setup process might be discouraging or confusing. To address this problem, we create a novel approach

for securely sending the network name and password to the unassociated wireless devices, which we call *Secure TRansfer of Association Protocol* (STRAP). STRAP allows a device to bootstrap its connection to the home wireless network with the help of a device already connected to the network (either through Ethernet or WiFi).

STRAP consists of three components, as shown in Figure 1: unassociated devices that are trying to connect, the home wireless router, and a device that helps the unassociated devices connect, which we call the "boot". STRAP requires no modifications to the home wireless router. The boot runs an application that securely transmits the network name and password, and the unassociated WiFi devices run an application to receive this information. The boot can be a device that the installer brings to the home (such as a laptop, smartphone or an Ethernet connected device) or an application a consumer installs onto one of their devices. The boot device must be able to inject Ethernet or WiFi frames into the network. We assume there is a trust relationship between the boot device and the devices trying to connect, and that both device types have been programmed with a installation ID and shared keys used for encryption and integrity protection. We expect this information to be loaded onto devices when flashed with firmware and prepared for an installation.

The major challenge we address in this paper is that there is no direct way to send secure information to the unassociated devices using just WiFi. We assume that all homes are using WiFi encryption (WPA, WPA2) and as a result, unassociated devices cannot decrypt the data without first knowing the network name and password. To make this task more difficult, since the boot device has an application running in userspace, it has no concept of 802.11 frames and at the lowest level can only send Ethernet frames. All data sent by this application will be encapsulated in an 802.11 frame and encrypted by the wireless adapter (if the boot device is wireless) or the wireless router (if the boot device is wired). To get around these obstacles, we create an approach where the boot device encodes data and places it in the source and destination address fields of an Ethernet frame, setting the destination address, such that the frame is broadcast to all wireless devices.

The first problem we must overcome is how an unassociated WiFi device can receive frames from a network it is not associated with. To solve this problem, we use monitor mode. Monitor mode is the mode of a wireless adapter that passes all received 802.11 frames for a specific channel, regardless of the intended destination and which network the frame is part of. This allows a WiFi device to receive frames, but if the network is encrypted, then the payload of the frame can not be read.

This leads to our second problem, how can a device in monitor mode receive unencrypted data? The key insight for this problem is that a wireless router does not encrypt the 802.11 header, which includes the destination and source addresses. This means an unassociated device in monitor mode can read the source and destination addresses of encrypted 802.11 frames. Using this fact, we can send encoded data in the source and destination addresses. One thing to note is that our application level code deals with Ethernet frames and the wireless routers encapsulates Ethernet frames into 802.11 frames. Luckily, wireless routers directly copy the Ethernet source and destination addresses and insert them into the source and destination addresses of the 802.11 frame [2]. This allows data to be sent by an already connected device (either through Ethernet or WiFi), through the home's wireless router, to an unassociated device in monitor mode.

The source and destination addresses make up 12 bytes in an Ethernet and 802.11 frame. This means at most, 12 bytes of data can be encoded into these two fields. Since an unassociated device can not transmit on a network, communication can only be in one direction, so the source MAC address does not need to be routable back to the boot device. However, care must be given as to how information is encoded into the addresses. The bytes in the MAC address have specific meaning that need to be followed or a frame risks getting rejected or colliding with other MAC addresses. The least significant bit of the first byte of a MAC address specifies whether the address is multicast (1) or unicast (0). The second least significant bit of the first byte specifies if the MAC address is globally unique (0) or locally administered (1) [3]. To follow this convention, we set these bits to `10` (locally administered and unicast). By doing so, we ensure that regardless of what data is being sent, the source and destination addresses will not collide with commercial MAC addresses.

Even with following these MAC address rules, arbitrary data can not be encoded directly into the destination address. A frame that is sent with a random destination MAC address will not be routable and will be dropped by the wireless router. The destination address needs to have the following two requirements. First, it needs to always be routable, regardless of what data is being encoded. Second, the frame needs to be transmitted on the wireless router's wireless interface so that the unassociated devices can receive the frame. To address these requirements, we use a key insight about special destination MAC addresses that have these properties: broadcast (`FF:FF:FF:FF:FF:FF`), IPv4 multicast (`01:00:5E:xx:xx:xx`) [4], and IPv6 multicast (`33:33:xx:xx:xx:xx`) [5], where the `x`'s of the addresses can be replaced with a unique identifier for that multicast group. These three addresses are always routable by the wireless router and always cause the wireless router to send the frame on all of its interfaces, including the wireless interface. For our purposes, all three addresses would accomplish the same goal of having the wireless router send the frame on its wireless interface. However, the IPv6 multicast address provides the most unused bytes, allowing us to encode more information in the address. For this reason, we select the IPv6 multicast address as the destination address, which leaves us with 10 bytes of usable data.

Since we are using the IPv6 multicast MAC address as the destination address, it is important to understand how other applications using this MAC address and STRAP will effect each other. To deal with IPv6 multicast traffic from other

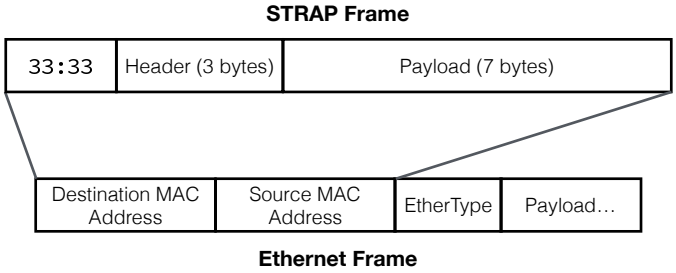

Fig. 2: The composition of a STRAP frame and how it fits in an Ethernet frame.

applications, STRAP uses an 6 bit ID field in its header (see the next section for details). This allows a device to uniquely identify IPv6 multicast traffic and ignore traffic that is not intended for it. STRAP would only affect another application using the IPv6 multicast address if the multicast group ID matches exactly with data STRAP is sending out. This effect would be minimized since the destination address of STRAP frames change between each frame because of packetization and encryption.

Using the the whole source address and part of the IPv6 multicast address as the destination address, we can encode 10 bytes of data inside each Ethernet frame. Figure 2 shows how a STRAP frame fits inside of an Ethernet frame. We recognize that the ability for an already connected device to send arbitrary data to unassociated wireless devices could have many different applications, such as emergency notifications, but for this paper we limit the usage to sending the network name and password. In the following sections, we outline the design of STRAP.

A. Header

Using the source address and IPv6 destination address as described above, there are 10 bytes that are available to encode data. We do not expect a network name and password to fit within 10 bytes, so we design STRAP to support fragmented data. We create a header format as shown in Figure 3. We use the first 3 bytes of the source address as a header for our protocol and the rest of the bytes as the payload. The first 6 bits are used as a unique ID to identify the install. Since each unassociated device will be in monitor mode, listening to all wireless communication on a certain channel, it is important that the device can filter out data that is unintended for it. Bits 6 and 7 set the MAC address to be locally administered and unicast, as mentioned above. Bit 8 is a flag used to distinguish between different groups of transmissions. The flag is alternated between one and zero, allowing a device to know when a group of transmissions has ended and another one has started. Bit 9 and 10 are used as an index into a predetermined array for how much erasure coding has been added. This is explained more in Section II-C. Bit 11 through 16 give the total number of packets being sent. With STRAP, we make sure there is always an even number of packets being sent, so the total number of packets must be shifted left by one. The

Fig. 3: STRAP header format

last 7 bits (bit 17 to 23) represent the sequence number of that packet. The rest of the 7 bytes are used as the payload to send the encrypted network name and password. One thing to note is that only 126 packets can be sent in one round of transmissions, due to the size of the total field in the header. This allows for 882 bytes of data to be transferred. This is not a problem for transmitting the network name and password, as the maximum size of a network name is 32 bytes and the maximum size of a password is 64 bytes [6]. If it were to become a problem, the information can be broken up into multiple exchanges and the data can still be transmitted.

B. Encryption and Integrity Protection

To ensure that the network name and password are protected, we use AES-128 with CBC [7] to encrypt the data. An initialization vector (IV) is generated and used to encrypt the data and is sent along with the encrypted payload. A "global" sequence number is also added to the payload. We use the term "global" to distinguish between the sequence number that is part of the header described in Figure 3. We use the current time in the form of an epoch timestamp as our global sequence number, since it is monotonic. Since the global sequence number is monotonic, it protects against replay attacks. A message authentication code is created based on the encrypted data and global sequence number. We use SHA256 as our hash for creating the message authentication code. This is added to the payload.

C. Erasure Coding

Erasure coding is the process of adding redundant data to the original data so that if there is data loss, the message can still be recovered. Data loss is an important consideration for STRAP. Since we are using the IPv6 multicast destination address, there are no link layer acknowledgements so our frames do not benefit from link layer retries. We use Zfec [8] as our erasure coding algorithm. Zfec has two parameters, k and m. m is the total amount of blocks that will be produced and k is the number of blocks needed to construct the original message. Adjusting m and k produces different size blocks of data. We select m such that the block size is equal to our payload size, 7 bytes. In order to decode the data from Zfec, the receiver must know k and m. m is included in the packet

header as the total number of packets. In order to save bits, we do not want to send k directly. Instead, we use an array of predetermined values and send an index into that array. The values in this array represent the maximum loss tolerated while still being able to decode the message. For example, the array [.2, .4, .6, .8] would represent 20% loss, 40% loss, 60% loss, and 80% loss. If the boot device wants to support 40% loss, it encodes the the data using Zfec such that $k = .4m$ and set bits 9 and 10 of each header to `01`. When a receiver receives a packet, it indexes into the array using bits 9 and 10, uses this value to calculate k based on m, and decodes the message. We design STRAP to support multiple erasure coding rates so that it can adapt to the environment it is running in. It allows STRAP to start with minimal erasure coding and increase if needed.

D. Running STRAP

Figure 4 shows the general flow of data through STRAP. To start the process of transferring the association information, the installer enters their WiFi network name and password into an interface on the boot device. The network name and password that are entered are first encrypted using the pre-shared encryption key and randomly generated IV. A message authentication code is created using a pre-shared integrity key, encrypted data, and a global sequence number. The IV, global sequence number, encrypted data, and message authentication code are concatenated together and run through the erasure coding algorithm. Finally, the data is split into packets, header bytes are added, and sent as empty Ethernet frames with only the source and destination addresses set. To ensure that all devices receive the data, the boot device will continue to repeat this procedure, toggling the send flag, updating the global sequence number, and creating a new IV.

Each of the unassociated wireless devices running STRAP enter into monitor mode and scan through all the channels, listening for frames from the boot device (using the ID segment of the header to filter out unwanted packets). If a device receives frames with different flag values, it knows a new transmission has started and discards the frames with the old flag. Once a device has received enough frames to complete the message, it authenticates the message (using the MAC and global sequence number), decrypts the message, and connects to the home wireless router. After a device connects to the network, it performs the necessary tasks to notify the boot device that it has connected, such as notifying a server that the boot device is connected to. As time goes on, the boot increases the amount of erasure coding it adds to the data. This helps to catch any devices that have high loss and are unable to decode the data. Once all of the expected devices have connected, the person installing the devices can stop the boot from sending the data. If the wireless network's name changes, STRAP can be rerun with the new network name and password. If a STRAP device is unable to connect to the programmed network, it automatically starts listening for STRAP frames.

E. Threat Model and Security Analysis

With STRAP, we assume that there is a trust relationship between the boot and devices trying to connect, and that two cryptographic keys have been loaded onto the unassociated devices being installed, before putting the devices in a person's home. One of the keys is used for encryption and the other is used for integrity protection. These keys have a similar role to the network name and password of the home's wireless network (to secure communication), but with the important difference that it does not have to be manually entered by a user or installer. The key is controlled by the company making the devices. The key could be loaded onto the devices when the devices are being flashed with firmware. The boot device can be preconfigured with the shared keys as well, or an installer can load the shared keys onto the boot device, for example, through a QR code or logging into an application.

Our adversary model includes both active and passive attackers. An adversary can eavesdrop on the conversation and replay previously captured frames or inject new frames, but has no physical access to the devices. Denial of service attacks, such as jamming, are beyond the scope of this paper. The goal of an adversary is to gain access to the home's wireless network or trick the connecting devices into connecting to a rouge access point. This could potentially give the adversary access to data the device produces.

We now outline some of the possible attack vectors. The transmitted data is encrypted using a shared secret, that only the boot and devices trying to connect know, so the adversary will be unable to decrypt the data. Since the same information will be transmitted multiple times, we update the IV between each round of transmissions to achieve semantic security. This ensures that an adversary listening to multiple rounds of transmissions will not be able to learn anything about the network name and password. The adversary cannot imitate the boot device, tricking the devices into connecting to a rouge access point, because it does not know the secret keys, so any data it creates and sends out will not pass the integrity check at the devices trying to connect. Assuming a participant has changed their network name and an adversary had captured the frames from this protocol when the participant was using the old network name, the adversary could set up an access point with the old network name and replay the frames it had captured. However, the connecting devices will detect this attack because of the global sequence number. The adversary is unable to change the global sequence number (or any of the encrypted data) without the devices detecting it, due to the message authentication code.

III. Implementation

We implement both the STRAP transmitter and receiver on Raspberry Pi 3s using Python. We have made the source code for STRAP publicly available [9]. For the STRAP receiver, we use the MediaTek MT7601 (Ralink 7601) wireless adapter which supports monitor mode. We connect the STRAP transmitter (the "boot" device) through Ethernet into the home access point. When the device running the STRAP receiver

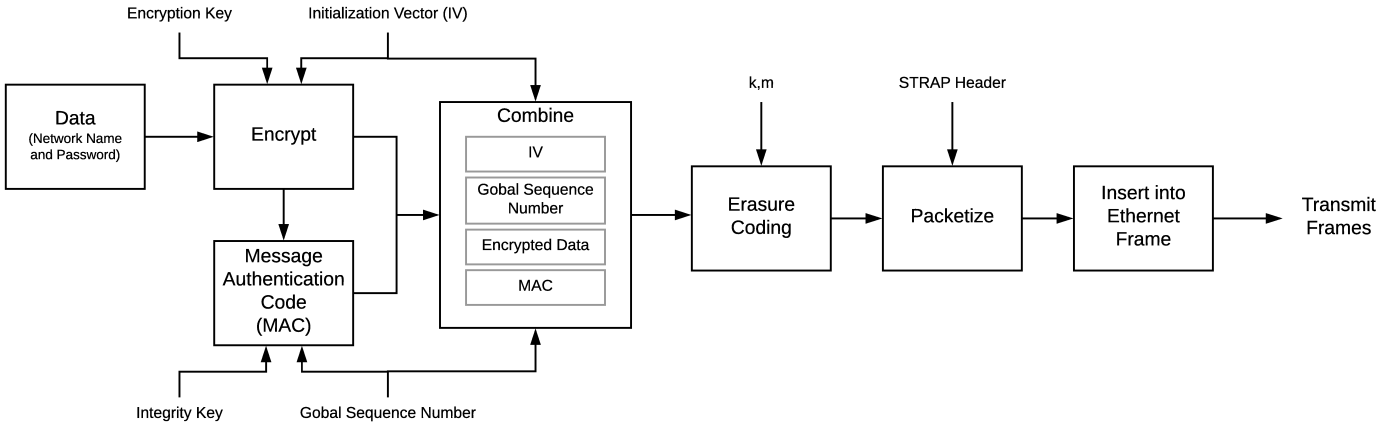

Fig. 4: Flow of data through STRAP

boots up, it looks for any previously known wireless network to connect to. If it is unable to find a network, it starts the STRAP receiver. The receiver randomly scans through the WiFi channels listening for STRAP messages, waiting one second on each channel. If the receiver detects a STRAP message, then it stops on that channel until it receives all of the frames necessary to decode the message. If the STRAP receiver is unable to receive any STRAP messages, it tries again to connect to any previously known networks. This process is continued until the device is able to connect to a network.

The STRAP transmitter is started by an installer entering the network name and password into a web page running locally on the boot device. The Python script uses the Scapy Python library [10] to inject Ethernet frames into the network. The STRAP transmitter continuously sends out frames every 50 ms until all of the expected devices have connected and the installer stops the boot device.

IV. Evaluation

We integrate STRAP with two air quality sensors, the Dylos DC1100 [11] and Plantower PMS3003 [12]. Both of these sensors are commercially available that measure the concentration of airborne particulate matter. STRAP has been used in homes to deploy these air quality sensors, with multiple sensors per home.

We evaluate four aspects of STRAP: hardware support, loss characteristics seen while using STRAP, effect of STRAP on normal network operation, and ease of use of STRAP compared to the commonly used temporary network solution.

A. Wireless Router Support

We test STRAP using five wireless routers in four locations to ensure that our protocol is supported across a broad range of brands. We test STRAP on an Apple Airport Extreme (4th generation), Google Fiber Network Box GFRG200, Netgear Nighthawk R7000, Linksys WRT3200ACM, and TP-Link WR940N. Of the five wireless routers we tested, all of them supported our protocol. Though we are overloading the purpose of the source and destination addresses of the Ethernet frame, we encode the data so that the source address should not collide with any other MAC addresses by setting the locally administered bit and the destination address uses to the standard MAC address for IPv6. As a result, STRAP works on all wireless routers we have tested.

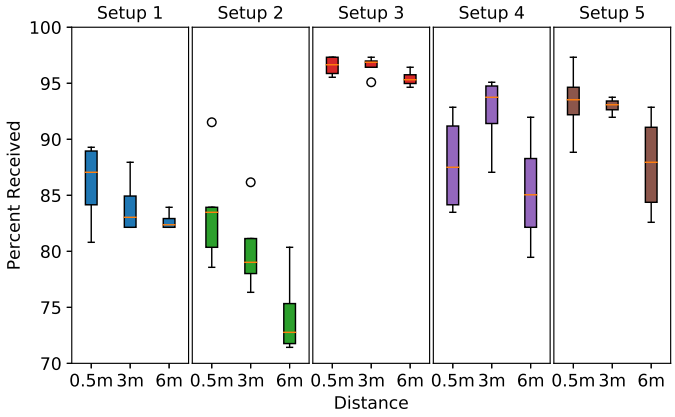

Fig. 5: Percent of frames received in different experimental setups (different wireless routers, different locations) vs. path length.

B. Loss Characteristics

We test STRAP at three different distances from a wireless router (less than .5 m, 3 m, and 6 m) to understand the loss patterns and help inform us how much erasure coding we need. The loss is important to understand since the 802.11 frames that are sent are multicast and do not benefit from link layer retransmissions. We use the same five wireless routers in the same four locations from the previous experiment. We run STRAP, measuring the number of packets that were received by a device trying to connect to the network. Figure 5 shows the results of these experiments. Each graph shows the results from one of the five access point setups mentioned in the previous section. A box plot is shown for each of the three transmission distances. The variations between the five setups is likely to be because of varying environmental conditions for

TABLE I: Network speed during normal network traffic and while running STRAP.

	Throughput (Mbps)	
	Average	**Std**
Normal Network	314.6	27.2
Using STRAP	335.2	9.6

TABLE II: Number of steps to configure devices using the temporary network method compared to STRAP.

Temporary Network Method	**STRAP**
Open smartphone WiFi settings	Open webpage or app (hosted on boot device)
Find name of device's temporary network and connect	Enter network name and password
Open application or website to interface with device	
Find network name from list and enter password	
Repeat steps for each device	

the same path length. The results show that the STRAP works well even from 6 m, with the lowest percent received being above 70%. This amount of loss can be overcome by using erasure coding.

C. Network Overhead

We measure the effect of STRAP on the other network activity. We do this by running `iperf` under normal network operation and while running STRAP. We run `iperf` on two laptops, one connected wirelessly and the other connected through Ethernet. Data is sent from the Ethernet connected laptop to the wirelessly connected laptop. Table I shows the results of these experiments. Although the average throughput while using STRAP was slightly higher, a two-sample t-test shows no statistically significant difference (at $\alpha = 0.05$) between `iperf` throughput using STRAP vs. not using it. In other words, use of STRAP does not affect the network's data rate performance.

D. Speed

We evaluate how long it takes a STRAP receiver to connect to a wireless router. The time it takes a STRAP receiver to receive the network name and password depends on how quickly the receiver is able to find the channel of the wireless router the boot device is connected to and capture enough STRAP frames to decode the network name and password. Assuming there are eleven WiFi channels for the STRAP receiver to scan through, the boot device is already transmitting, and the STRAP receiver pauses on each channel for one second (see Section III), it will take at least 1 second and at most 11 seconds.

The time it takes to transmit all of the STRAP frames depends on how large the combined network name and password are and the amount of erasure coding used. We look at the best and worst case scenario for STRAP. In the best case, assuming a small network name and password and no loss on the network, it would take only 11 out of 14 frames to be received for the receiver to get the network name and password. With STRAP, a frame is transmitted every 50 ms, so it would take 550 ms to receive the first 11 frames.

In the worst case, it would take 120 frames to be sent, the maximum number of frames STRAP supports, and only 24 frames received (80% loss). Note, to send the maximum number of frames, erasure coding would need to be set to 80% loss and the combination of network name and password would need to be 111 characters long! As stated before, STRAP transmits frames every 50 ms, so it would take 6 seconds to receive all the necessary frames. However, for a STRAP receiver to catch at least one full transfer of those frames, the receiver needs to be on that channel for twice that time, 12 seconds. Combining the best and worst cases with the time it takes the scan the WiFi channels, STRAP can range from 1.55 seconds to 23 seconds. This time stays constant regardless of how many devices are being connected. The time taken for other approaches, such as the temporary access point method, increases with the number of devices that a person is connecting.

Next, we compare the number of steps required to set up a device using the standard method of having the device become a temporary network compared to using STRAP. The steps are shown in Table II. Using the temporary network method, four steps are needed for *each device* that is set up. STRAP only requires two steps in total regardless of the number of devices that are being set up.

Based on these evaluations, we conclude that STRAP will work with a home's wireless router, work from a variety of distances, has no effect on a home's network while running, and will make it easier to install devices than traditional methods.

V. Alternative Approaches and Related Work

The deployment scenario we explore is that an installer from a company providing an IoT system will come to a home with many devices needed to be installed. Alternatively, the company may simply mail a set of devices with instructions, and the user serves as the installer.

Given that we assume there is a trust relationship between the person installing the devices and the devices themselves, and that each device has to be programmed with a shared key, it would make sense to just program the devices with the home's WiFi network name and password instead of the shared key before installing the devices in the home. However, this approach has drawbacks. First, a resident might be uncomfortable giving out their WiFi password, and an installer might not want to be responsible for storing it. Second, there can be a high rate of errors when communicating the network name and password in part due to default long and difficult passwords on home wireless routers. These passwords may be challenging to relay accurately (e.g., mistaking a capital letter "o"). An incorrect network name or password is only detected

after installing the devices. In this case, the installer would need to reprogram the devices with the correct information. Depending on the device, this might not be possible to do in a home or it might take a long time.

Another solution would be to use the boot device as an access point itself. Devices trying to connect could be configured to automatically connect to the boot device's network and then the boot device could share the network name and password with the devices. However, this approach requires that the boot device can become an access point which limits the types of hardware supported. In a self-installation setting where a customer is using their own device as the boot device, this might not be possible.

One possible solution to the WiFi bootstrapping problem is to use an alternative WiFi security model. WPA Enterprise is an alternative to the traditional residential security method, WPA2. Instead of providing a network name and password, a device connects to the access point and provides a username and password. A RADIUS authentication server authenticates the user allowing the device to connect to the network or rejects the device, disconnecting it from the network. This requires the device that is trying to connect to have three pieces of information: network name, username, and password. A home could create a username and password for the devices that are going to be installed, alleviating the problem of having to program the devices with this information. However, the network name would still need to be programmed in and if the network name changes, the devices would have to be reprogrammed again. Another alternative would be to set up a wireless network with no security (open) and instead use a captive portal to authenticate users. This solves the problem of having to program a device with a password, though it creates new problems with not having the transmitted data encrypted. Both of these approaches are not features supported by standard consumer wireless routers and would require advance configuration to setup. Expecting a home to support them in order to install devices in the home is not acceptable. STRAP works without any modifications to the home wireless router.

Another way to circumvent this problem is to not depend on a home's WiFi at all and instead use cellular connectivity. However, using cellular can be cost prohibitive and might not be an option for some device types or deployments. For this reason, we focus on improving WiFi bootstrapping.

Many approaches exist to bootstrap the wireless connectivity of a small device (one without a screen or keyboard) to an wireless router. We outline some of the approaches used by current IoT devices. A common approach is for the IoT device to create a temporary access point, allowing the user to connect and program the network name and password onto the device. This approach is taken by many companies, such as Amazon for their Echo devices [13], and is a standard feature for many WiFi SoC, such as the Espressif ESP8266 [14] and TI CC3200 [15]. Typically, when a device first boots up and is unable to connect to a network, it enters into access point mode. The user is then prompted to connect to the temporary network, usually through a smartphone app. Once connected, the user enters the network name and password, usually through an app, which programs the device. The device switches from access point mode into client mode and connects to the network. While this approach works well for a few devices, it does not scale when dealing with many devices. Having to enter in a network name and password quickly becomes cumbersome and error-prone, especially when deploying multiple devices in a single installation. This approach also requires some kind of smartphone application to guide a user through the setup and enter the WiFi credentials into.

Another approach would be to use WiFi Protected Setup (WPS) [16]. The promise of WPS is to simplify connecting a new device to a WiFi network. This allows a user to have a secure network but not have to enter a password into the device. WPS has two modes: PIN and push button. With PIN mode, a person enters a PIN into the device that wants to connect. The PIN comes from the wireless router, usually on the sticker on the bottom of the wireless router. With the second mode, push button, a button is pushed on the wireless router and on the device that wants to connect. Information is exchanged between the two entities, allowing the device to connect to the network. While WPS seems to address this bootstrapping problem, there are drawbacks with the implementation. First, the PIN mode of WPS suffers from online and offline brute force attacks [17]. As a result, it is recommend by security experts to disable WPS [18]. Second, WPS is *not* part of the WiFi specification, and as a result, not all wireless routers support it. Since both the home's wireless router and devices trying to connect must support WPS to work, not having full support makes this approach infeasible. STRAP does not depend on a feature of the wireless router to work and instead uses basic Ethernet frames.

Another option is to use an out-of-band channel for communicating the network name and password. Some commercial devices, such as Google Home, use Bluetooth for the purpose of receiving WiFi association information [19]. Other commercial devices, such as the Amazon Dash Button, use "ultrasound" to transmit data between a smartphone and device [20]. With Bluetooth, a person is only able to program one device at a time. With both methods, these approaches require extra hardware and are tightly coupled with a smartphone application. Our approach does not require any extra hardware on the device or a smartphone application.

STRAP overloads the purpose of the source and destination addresses of the Ethernet and 802.11 frames to encode data. Similar approaches are used for covert channels. WiFi covert channels mainly focus on two main methods for conveying information, timing and the 802.11 header. WiFi timing covert channels encode information in the timing between frames [21] [22]. Covert channels using the 802.11 header overload the purpose of a field to transmit data, such as STRAP does with the source and destination addresses. Such work includes modifying the sequence control and fragment control fields of the 802.11 MAC header [23] or rate switching [24]. Other covert channels focus on modifying IP and TCP fields

that are not typically used or not checked by devices [25].

VI. Conclusion

In this paper, we have discussed the motivation, design, implementation, and evaluation of STRAP. STRAP allows for multiple devices bootstrap connectivity to a home's wireless network at one time. It is a novel design that encodes data into the source and destination addresses of Ethernet frames. Our evaluation shows that STRAP works with a variety of routers and has no effect on the network when running. We also show that STRAP is much faster compared to conventional approaches to connecting devices to a home's wireless network.

Acknowledgments

Research reported in this publication was supported by NIBIB of the US NIH under award number 1U54EB021973-01.

References

[1] A. E. Guttmacher, S. Hirschfeld, and F. S. Collins, "The national childrens studya proposed plan," *The New England Journal of Medicine*, vol. 369, no. 20, p. 1873, 2013.

[2] WARP: Wireless Open-Access Research Platform. (2014) Ethernet encapsulation and de-encapsulation. [Online]. Available: https://bit.ly/2rvJJ9X

[3] I. S. Association. (2011) Standard group mac addresses: A tutorial guide. [Online]. Available: http://standards.ieee.org/develop/regauth/tut/macgrp.pdf

[4] S. Deering, "Host Extensions for IP Multicasting," Internet Requests for Comments, RFC Editor, RFC 1112, August 1989. [Online]. Available: https://tools.ietf.org/html/rfc1112

[5] M. Crawford, "Transmission of IPv6 Packets over Ethernet Networks," Internet Requests for Comments, RFC Editor, RFC 2464, December 1998. [Online]. Available: https://tools.ietf.org/html/rfc2464

[6] "Ieee std 802.11-2012 (revision of ieee std 802.11-2007)," pp. 1–2793, March 2012.

[7] U. S. N. I. of Standards and T. (NIST). (2001) Announcing the advanced encryption standard (aes). [Online]. Available: https://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.197.pdf

[8] L. Rizzo, "Effective erasure codes for reliable computer communication protocols," *SIGCOMM Comput. Commun. Rev.*, vol. 27, no. 2, pp. 24–36, Apr. 1997.

[9] P. Lundrigan. (2017) Unassociated transfer. [Online]. Available: https://github.com/philipbl/unassociated_transfer

[10] secdev. (2018) Scapy. [Online]. Available: http://www.secdev.org/projects/scapy/

[11] Dylos, "Dc1100 air quality monitor," 2018. [Online]. Available: http://www.dylosproducts.com/ornodcairqum.html

[12] Plantower. (2018) Pms 3003-pm2.5-plantower technology. [Online]. Available: http://www.plantower.com/en/content/?107.html

[13] Amazon. (2018) Connect your echo device to wi-fi. [Online]. Available: https://amzn.to/2m6Cohx

[14] Espressif. (2018) Esp8266 overview. [Online]. Available: https://www.espressif.com/en/products/hardware/esp8266ex/overview

[15] T. Instruments. (2018) Cc3100/cc3200 simplelink wi-fi internet-on-a-chip. [Online]. Available: http://www.ti.com/lit/ug/swru368a/swru368a.pdf

[16] W.-F. Alliance, "Wi-Fi Protected Setup," https://www.wi-fi.org/discover-wi-fi/wi-fi-protected-setup, 2017.

[17] S. Viehbck. (2011) Brute forcing wi-fi protected setup. [Online]. Available: https://sviehb.files.wordpress.com/2011/12/viehboeck_wps.pdf

[18] J. Allar. (2012) Vulnerability note vu#723755 - wifi protected setup (wps) pin brute force vulnerability. [Online]. Available: https://www.kb.cert.org/vuls/id/723755

[19] Google. (2017) Set up your google home device. [Online]. Available: https://bit.ly/2KLc4Sr

[20] M. Gibbs. (2015) Hacking amazon's dash button. [Online]. Available: https://bit.ly/2jIZKG7

[21] R. Holloway and R. Beyah, "Covert dcf: A dcf-based covert timing channel in 802.11 networks," in *2011 IEEE Eighth International Conference on Mobile Ad-Hoc and Sensor Systems*, Oct 2011, pp. 570–579.

[22] R. Archibald and D. Ghosal, "A covert timing channel based on fountain codes," in *2012 IEEE 11th International Conference on Trust, Security and Privacy in Computing and Communications*, June 2012, pp. 970–977.

[23] L. Frikha, Z. Trabelsi, and W. El-Hajj, "Implementation of a covert channel in the 802.11 header," in *2008 International Wireless Communications and Mobile Computing Conference*, Aug 2008, pp. 594–599.

[24] T. E. Calhoun, X. Cao, Y. Li, and R. Beyah, "An 802.11 mac layer covert channel," *Wireless Communications and Mobile Computing*, vol. 12, no. 5, pp. 393–405, 2012. [Online]. Available: http://dx.doi.org/10.1002/wcm.969

[25] S. Zander, G. Armitage, and P. Branch, "A survey of covert channels and countermeasures in computer network protocols," *IEEE Communications Surveys Tutorials*, vol. 9, no. 3, pp. 44–57, Third 2007.