Level of Constraint Eviction Policy to Reduce DTLS Handshakes for Constrained Devices

Peter Yurkovich
Civil Engineering
Virgina Tech
Blacksburg, USA
peteryurkovich@vt.edu

Irena Shaffer
Computer Science
Virginia Tech
Blacksburg, USA
irenas4@vt.edu

***Abstract*—The Internet of Things has become increasingly popular with more and more devices being built and connected to each other and the Internet. This paper gives an overview of some of the challenges and constraints of security in IoT, the technical approaches and policy standards for security in IoT applications, and some of the open research questions in IoT security. This paper then proposes a novel solution of maintaining communication connections with highly constrained devices to reduce the number of times highly constrained devices have to perform the computational complex DTLS Handshake used in CoAP. This is done using two eviction policies based on the resource constraints of an IoT device. These eviction policy are then compared to other common eviction policies for different scenarios.**

***Index Terms*—IoT, CoAP, DTLS Handshake, Eviction Policy, Connection Management**

Members' roles in the project:

Peter Yurkovich: Research on technical approaches and policies in IoT with specific focus on CoAP. Conception of novel idea and overall simulation setup and design. Development of Level of Constraint byte. Programming scenarios, LRU, CO, and CSE eviction policies. Running simulations and compiling results. Created the results visualization.

Irena Shaffer: Research on challenges and constraints in developing security for IoT. Research on open research issues in IoT security. Research on general technical approaches and policies in IoT. Research on currently used eviction policies. Conception of novel idea and overall simulation setup and design. Programming scenarios and LFU eviction policy. Running simulations and compiling results.

I. Introduction

The Internet of Things (IoT) refers to the vast number of devices and sensors that wirelessly communicate with each other and the internet. While the use of IoT devices has become more popular lately, there is no set start to the use of IoT devices and networks, but rather was something that naturally evolved over time [1]. Starting in the late 1960's computers began to be able to communicate using computer networks. The invention of the TCP/IP stack and the World Wide Web in the 1980's and 1990's led to a boom in the popularity of the internet. As this popularity grew and mobile devices began to appear, the concept of the Internet of Things was developed. Now billions of devices are part of this IoT and the number of devices is rapidly increasing.

Some examples of IoT devices are smart sensors, smart home security devices, wearable health monitors, smart factory equipment, autonomous farming equipment, and smart meters. IoT has many different applications. On example is smart homes where many settings of the house such as air conditioning, ventilation, and heating can be controlled remotely. Sensors can detect occupants of the house and change certain settings automatically. Appliances such as the refrigerator, dishwasher, and washing machine can also be controlled remotely. Many smart homes also have automatic security features. IoT devices also have medical applications. Medical trackers placed on or near people can detect early signs of health anomalies and inform the user to seek medical attention. Smart cities use a variety of IoT devices such as sensors that monitor traffic, devices that control lighting, surveillance cameras, and environmental monitoring, to make improvements to the city.

Generally IoT is divided into three layers: the perception layer, transportation layer and the application layer. The perception layer gathers data. The transportation layer transmits data collected by the perception layer. The application layer consists of solutions and applications that interact with the user [2]. In addition to these layers, the IoT protocol stack is defined using the physical (PHY) layer, Medium Access Control (MAC) layer, adaption layer, network layer, transport layer, and application layer [3]. An example of a standard IoT network stack is shown in Fig. 1.

II. Challenges and Constraints

One of the largest challenges within typical IoT networks is that IoT devices are often resource constrained. Many devices run on low power provided by batteries or alternative sources. Most IoT devices do not run full operating systems [5], making many current security solutions for computers infeasible. In addition, many devices have limited memory and computing power. Thus, security solutions for the IoT must be energy efficient and cannot require extensive computations [2]. These factors make current cryptographic algorithms that are generally used for security impracticable for use in the IoT.

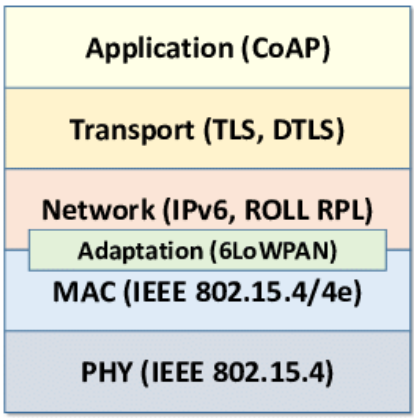

Fig. 1: Visual representation of a standard IoT network stack [[4]].

Traditional security and enforcement mechanisms are ineffective due to the nature of the IoT [5]. Static perimeter network defenses, such as firewalls and intrusion detection/prevention systems, do not work because IoT device behaviors often change and these devices are embedded deep inside these networks. End-host based defenses, like antivirus solutions, often will not work in the resource constrained environment of IoT devices. Another common security solution is software patches. While these can be effective in fixing security flaws, there are some issues for IoT devices. First, it is often difficult to get the patches to all the affected devices. Also, many devices will continue to be used after the vendor has stopped making and selling these devices. So, the vendors stop making patches and the devices are left with security flaws. In addition, many devices, especially sensing devices, must be placed in locations that are easily accessible, putting these devices at risk of physical tampering.

Most vendors are more interested in producing new products and services than spending time working on security for these devices because of the increasing popularity and demand for IoT devices. This leads to networks containing many nodes with little or no security. Since these devices have access to the network it is possible for a compromised node to allow an intruder into the network. Additionally, the large number of nodes in the system makes the system vulnerable to denial of service attacks.

Traditional methods of learning signatures and anomalous behaviors are unlikely to work for IoT devices because of the diversity and inter dependencies of devices in the IoT [5]. One of the biggest challenges, is detecting security issues in implicit dependencies between devices. Implicit dependencies are cases where devices do not directly communicate but are linked somehow though the physical environment. The paper by Yu, Sekar, Seshan, Agarwal, and Xu gives the example of an implicit dependency in devices in a smart home that can lead to a security flaw [5]. In this smart home example there is a thermostat that controls the air conditioning, a temperature sensor, and windows that are opened automatically if the air conditioning is off and the temperature reaches a certain threshold. If an attacker can turn off the air conditioning, the temperature will rise, and the temperature sensor will signal that the windows should open. This now allows physical access to the house. Standard solutions for detecting anomalous behavior is not designed to deal with these types of inter-dependencies.

III. Protocols and Technical Approaches to Security in IoT

The standard for security at the PHY and MAC layers is given by the IEEE 802.15.4 protocol [6]. This standard is designed to optimize the energy efficiency though a trade-off with the rate of data being sent. Communications are secured using the Advanced Encryption Standard (AES). IEEE 802.15.4 designates 8 different modes of security with different levels of encryption and authentication [3].

At the adaption and network layer, the IETF IPv6 over Low-power Wireless Personal Area Networks (6LoWPAN) specification is used for transportation of IPv6 packets over low-energy IEEE 802.15.4 [3]. There are currently no standard security mechanisms defined for 6LoWPAN but there are some proposals and research being done in this area. Routing in 6LoWPAN environments is defined by the Routing Protocol for Low power and Lossy Networks (RPL) Protocol [7]. RPL is adaptable to different application areas. The current RPL specification uses AES with 128-bit keys, and RSA for digital signatures. RPL has three security modes: unsecured, preinstalled, and authenticated.

In the application layer there are a few standard protocols that are widely used in IoT devices. These include Message Queue Telemetry Transport (MQTT), Extensible Messaging and Presence Protocol (XMPP), and Constrained Application Protocol (CoAP).

MQTT is an asynchronous publish/subscribe application layer protocol that utilizes the TCP stack to target lightweight Machine to Machine (M2M) communications [8]. The MQTT protocol is optimized to minimize bandwidth and battery usage, making it ideal for IoT devices. In the MQTT protocol there is a sever that contains different topics. Each client can be a publisher and/or a subscriber that sends information to the server about a topic and/or receives messages when there is an update to a topic it is subscribed to. Security for MQTT is done using username/password authentication which is handled by both TLS and the Secure Sockets Layer (SSL).

The XMPP protocol is designed for chatting and message exchange [8]. It runs over TCP and provides asynchronous publish/subscribe messaging systems as well as synchronous request/response messaging systems. XMPP is extensible and XMPP Extension protocols can be used to increase its functionality. One downside of the use of XMPP in IoT devices is that it used XML messages which have additional overhead and increased power consumption. TLS/SSL security protocols are built in to the XMPP specification.

CoAP is an internet protocol specialized toward constrained node and network usage [9]. CoAP was written under the

assumptions that it would be used by devices with only 8-bit microcontrollers and on networks with high packet error rates, and low data throughput, such as 6LoWPANs. CoAP utilizes UPD and the Constrained RESTful Environments (CoRE) Link format, which is a modification on the traditional REST architecture focused on constrained devices and networks. The primary security measure behind CoAP is the use of Datagram Transport Layer Security (DTLS). The novel security solution described in this paper is for use in the DTLS handshake protocol used in CoAP, thus this section will give a more detailed overview of CoAP and DTLS protocols.

CoAP utilizes a 4-byte header, followed by options, and then a payload. There are four base types of CoAP messages, Confirmable (CON), Non-confirmable (NON), Acknowledgement (ACK) and Reset (RST). Confirmable messages are those which need reliability and will broadcast until either an ACK or RST response is emitted from the server, or until a maximum number of retransmits is reached. NON messages are not required to be responded too, but if a recipient is not able to process it, they may send an RST message in response.

The CoAP option field contains 3 required parts, and 2 optional ones [10]. The first field is the option number. The option number is a 4-bit option delta which is the difference from the previous option number, or the difference from 0 in the case of the first option. The second field is a 4-bit option length, which specifies the length in bytes of the final required field, the option value. The two optional parts are an extended option delta and an extended option length which allow for larger numbers to be used than the above described 4-bits of space. Both the option delta and option length reserve the numbers 13-15, with 13 and 14 indicating the usage of the extended fields.

There are four types of options included in CoAP messages: critical options, elective options, unsafe options, safe-to-forward options. Critical options are options which require the understanding of an option to process a message. Endpoints which do not understand the options are expected to lead to an error request or reject the message. An elective option is an option which can be ignored by endpoints which do not understand it. If an endpoint does not understand the option, it can still process the message. An unsafe option is an option which requires any proxies to understand the option in order to forward it. Conversely, a safe-to-forward option is one which does not require a proxy to understand the option in order to forward it.

Three of CoAP's most powerful, but security concerning, tools are proxying, caching, and multi-cast. CoAP introduces two types of proxy nodes which are able to resend or respond to messages without being the server. A Forward-Proxy is a proxy which was selected by the client to forward messages, while a Reverse-Proxy is one that is able to do the work without informing the client that the Reverse-Proxy is not in fact the server. A Forward-Proxy may be required to translate any requests into what will need to be submitted to the server, however a Reverse-Proxy needs to receive the requests as they would be submitted to the server. Proxying is a large concern for security because by definition, it is a man-in-the-middle, thus it can allow unprivileged users to access the data being sent and received. In order to quicken the sending of data and remove already completed computation, caching messages and responses is common for both proxies and servers. Caching messages can be security concerning since caches shared or accessed by multiple devices can allow insecure access to data that was previously sent securely.

Multi-cast is a key feature allowing devices to collect data from multiple sensors at once. Since there may be a large number of devices all of which perform the same task, ie. temperature or weather nodes, it is imperative for some networks to implement this despite it not being supported by DTLS. Nodes may assign themselves to group addresses, and then listen in and respond to requests to that address. Multi-cast requests must be non-confirmable, and if a server recognizes that the message is multicast, it must not send back an RST message.

There are four security modes within CoAP: NoSec, PreSharedKey, RawPublicKey, and Certificate [9]. Within NoSec, no protocol security is used, however other lower level security options such as IPsec can be used. Outside of NoSec, DTLS is the default security for CoAP. This method's security is only able to come through the prevention of attackers from sending and receiving packets in the network. In PreSharedKey, keys are distributed before the nodes are deployed between whichever nodes will need to communicate. This method prevents secure transmissions from occurring between newly created or discovered nodes. As such, this should only be used when a set plan is completed for a IoT network ahead of time. RawPublicKey uses DTLS and provides each device with an asymmetric key pair. These key-pairs are validated using an out-of-band mechanism. For example, if the main communication mode is the wireless communication, then SMS or a temporary wired connection might be used to validate the key pairs for newly connected devices. The final alternative is Certificate, within which a x.509 certificate is used and signed according to a root authority and is used to complete a handshake process.

Since responses will often contain amounts of data significantly larger than the requests, amplification attacks can be launched when NoSec is used. The usage of the well-known URI from CoRE is an easy way for attackers to gain access to the size of the data accessible at a node, which can allow them to request the largest objects available from a node [11]. If a large number of significantly sized responses are created and sent, it can act as a DoS attack on a targeted server, or on the IoT network itself. Since less constrained devices on the network will often support cross-protocol between CoAP and HTTP, NoSec CoAP networks can be used to create DoS attacks outside of their own network as well. By reducing the amount of data that can be responded with from one request, the total amount of data congesting a network or an external server being accessed by HTTP can also be reduced. Multi-cast requests can also be a source of a DoS attack, and as such multi-cast requests should only be accepted when the

requests can be authenticated in some way, in addition to the preventative measures of the the previously mentioned Leisure time.

Spoofing attacks within NoSec or PreSharedKeys could attack singular nodes, groups of nodes, or the entire network. Spoofing RST, ACK, response messages, and multi-cast messages can all have significant and unique impacts on the network and nodes. Spoofing attacks can help be mitigated through the randomized token in the request.

In IoT and constrained nodes, all modes of DTLS may not be applicable [12]. The initial handshake and message overhead may be too excessive for the constrained node to be able to handle. Due to the initial handshake, after DTLS connections are opened, they should be held open for a long as possible, since the overhead of the handshake makes recreating a connection have a very high cost. Within requests and responses, the DTLS session and epoch must remain the same. If the initial handshake and message overhead is too high for the system, then the Certificate option for CoAP would not be an option. As such, each network and set of nodes will need to be analyzed to determine what the best possible form of security available is.

IV. Open Research Questions

One of the biggest challenges with IoT security is the large number of devices and the different implementations and uses of these devices [13]. A proposal to this problem is to treat the entire system of IoT devices as a single entity and find security solutions for the whole system instead of a single device, piece of software, or IoT layer. One proposal for this kind of solution is to find the similarities of different security architectures and create an abstract security architecture that provides all the basic security for IoT devices. This would behave like an abstract class or interface in software engineering. Specific security solutions for a type of device meet the criteria described by the extraction but can implement it in different ways and add additional security. Thus the security solutions for different devices are compatible but do not have to be the same.

Due to the number and diversity of devices in the IoT, approaches that are generally used to discover anomalous behavior and attack signatures cannot be used [5]. To increase the complexity of this issue, IoT devices are often cross dependent. These devices can communicate directly like traditional devices. However, one device can also cause changes to the environment that cause another device to react in a certain way. Attacks that target this indirect communication are different from traditional attacks to normal approaches to detect attacks will not work in this situation. Thus, this is an open security issue that will need to be handled as smart home-like applications become more popular.

Many IoT devices are sensors that collect large amounts of very different types of data [13]. Because of the size and diversity of this data and the constraints of IoT devices this data must be handled and organized efficiently. Keeping all of this data secure is a challenge that requires more research. It is possible that solutions for big data on the internet may be applicable here.

Security of communications in IoT is an area with many open research questions [3]. Starting with the network layer, some of the main areas of open research questions are maintaining the confidentiality, integrity authentication, and non-repudiation of messages, and key management [14].

In order to maintain confidentiality, integrity authentication, and non-repudiation, one proposed solution that has not yet been implemented is developing end-to-end network security so that messages that have to travel between nodes of an IoT system will remain secure [15]. Due to the difficulty of adapting current security mechanisms to the low power environment of the IoT, a few research proposals exist but have not been implemented. Many of these proposals involve the use of compressed security headers to allow current security protocols to be adapted to the IoT.

Another open research issue, in application layer security, is developing approaches for support of public keys and digital certificates [3]. Some proposed solutions include certificate pre-validation and session resumption in order to reduce the impact on the resources of IoT devices.

Key management is an issue for many IoT devices due to their small memory and computing power. Thus one open research question is how to make lightweight key management solutions. The process of key generation and exchange, key bootstrapping, in IoT devices is currently an area of interest for researchers . Some areas of interest to improve key bootstrapping in IoT are: optimized asymmetric protocols, blockchain technology, hardware-based and post-quantum cryptography [16].

Asymmetric protocols based on public key are not applicable to IoT devices due to the large key size [16]. Scaling down the key size for these protocols makes them insecure and would require additional authentication mechanisms for secure key exchange. Currently most certificate-based methods are too large and need to be optimized for the IoT. Some work has been done to compress DTLS headers and on compression of X.509 certificates. Future work can be done to make compressed certificates compatible with existing internet certification infrastructure.

Hardware assisted cryptography uses a Trusted Platform Module alongside DTLS and RSA to assist with asymmetric cryptography [16]. The Trusted Platform Module is an embedded chip that stores keys and efficiently performs cryptography operations. While this improved the efficiency and performance of the system it is unclear if this form of hardware assisted cryptography can feasibly be implemented on a large number of devices and makes this an area of study on both the hardware and software sides.

Additional areas of research are in making DTLS for use in CoAP less computationally expensive. The DTLS handshake, which is used for authentication and key agreement, is often too resource intensive for the devices in the IoT. Thus there is interest in modifying DTLS to make it more compatible with IoT devices. Some proposals for this include compression of

DTLS headers [17], using CoAP to support DTLS handshake operations [4], offloading some of the costly DTLS operations onto more powerful devices. This paper proposes a method for reducing the number of DTLS handshake operations required by highly constrained devices, thereby allowing a greater amount of constrained devices to use DTLS and increase security.

V. Proposed Solution

The DTLS handshake operation to initialize communication between two devices is much more computationally costly for resource constrained devices than maintaining the communication session between the devices [18]. Thus one way of reducing resource costs for IoT devices is to maintain communication sessions between devices as long as possible. However, these connections still need some memory to be maintained, so there is a limit to the number of connections a device can maintain at one time. When a new connection is added a different communication session needs to be closed. There are a variety of eviction policies that are used to determine which connections to close, but most of these based on the time a connection has been open or the frequency of messages sent over a connection.

There are currently many eviction and cache replacement policies that are used [19], [20]. Some of the most common are Random Replacement (RR), First In First Out (FIFO), LRU (Least Recently Used), and LFU (Least Frequently Used). RR randomly chooses an item to remove. Generally RR performs worse that other policies however it is easier to implement and less computationally expensive than many other policies. With the FIFO policy, the oldest item is replaced with the newest on the assumption that communications that have been open for a long time are no longer needed. This method is easy to implement and computationally efficient. The LRU policy removes clients that have spent the longest time since sending a message to the server. Generally, LRU performs better than FIFO but it is a little more complex to implement. LFU removes the least popular client. The server keeps track of the number of messages sent by each client and removes the one that has sent messages least frequently since opening the connection. While this method is often more effective than other methods it is more computationally expensive. Some additional policies that have been used are Size, where largest objects are removed first, and Most Recently Used (MRU). A relatively new method that has been proposed is Predictive Eviction, where the server knows when the client will send new messages and uses this to determine which client to evict [21]

There are different scenarios where some eviction policies are more effective than others.In cases where many messages are sent over a short period of time followed by long periods of no activity FIFO and LRU perform well but LFU performs poorly because popular clients remain connected even after they stop sending messages. In cases where different clients send messages at semi-uniform time intervals but at different frequencies LFU performs well but FIFO does not because clients that send messages a low frequencies remain connected longer then they should. One case where all the time and frequency dependent policies perform very poorly is cyclic uniform sending time where all devices send messages at the same set interval of time since all nodes are only able to send one message before being replaced by another node.

Our proposed solution contains two novel Constraint Based Caching (CBC) mechanism. The first is Constraint Only (CO), in which each cache will ensure that nodes with high levels of constraints are not evicted over those with low levels of constraints. We also propose Constraint Split Eviction (CSE), where one set of the cache is dedicated to constraints, and the other to a LFU cache structure. Within a CBC cache, Time To Live (TTL) must be considered as the cache could grow stale if old items are not removed. The TTL value must be greater than the length of any individual SEP sleep period. Within CSE the purpose of the non-constrained region, is to ensure that the most use is extracted out of those that are staying cached. This is accomplished through the usage of the LFU algorithm. Within the split cache, any new objects check first to see if any of the caches are stale. If none are, the new connection compares itself to the least-constrained item in the cache. If it is more constrained, then it evicts the least-constrained and takes it's place. If it is less constrained, then it evicts the worst item in the LFU cache, and then takes it's place.

For the CBC eviction policies to work, the server must be able to determine which client is the most constrained. We propose to do this using a Level of Constraint CoAP option. Constrained devices can be classified based on their computing power (Cx) (Table I), energy limitations (Et) (Table II), and strategy of using power for communication (Ps) (Table III) [22].

Name	Data Size (e.g., RAM)	Code Size
Class 0, C0	$<<$ 10 KiB	$<<$ 100 KiB
Class 1, C1	$\sim$ 10 KiB	$\sim$ 100 KiB
Class 2, C2	$\sim$ 50 KiB	$\sim$ 250 KiB

TABLE I: Classification of constrained devices based on computing power [22]. (KiB = 1024 bytes)

Name	Type of Energy Limitation	Example Power Source
E0	Event energy-limited	Event-based harvesting
E1	Period energy-limited	Battery that is periodically recharged or replaced
E2	Lifetime energy-limited	Non-replaceable primary battery
E9	No direct quantitative limitations to available energy	Mains-powered

TABLE II: Class of energy limitations [22].

Since each of the groupings contains mutually exclusive events, each device can be represented by three values one from each grouping. Each value can be represented using two bits as shown in Table IV. These bits can be stored in a single Level of Constraint byte as shown in Figure 2. This byte can

Name	Strategy	Ability to Communicate
P0	Normally-off	Reattach when required
P1	Low-power	Appears connected, perhaps with high latency
P9	Always-on	Always connected

TABLE III: Strategies of using power for communication [22].

be included in the handshake process allowing for the caching to be completed with full information.

Name	Bit Definition
P0	00
P1	01
P9	10
E0	00
E1	01
E2	10
E9	11
C0	00
C1	01
C2	10

TABLE IV: Definition of bits for Level of Constraint byte

The proposed solution would feature the Level of Constraint byte as another CoAP option which can be used in the handshake process. This would be an elective, safe-to-forward option. Since all of the option data has been compressed down to one byte the entire option would be contained in either 2 or 4 bytes depending if the Option Delta (extended) is needed to be used or not.

0	1	2	3	4	5	6	7
0	0	Ps		Et		Cx	

Fig. 2: Visual representation of the Level of Constraint byte

The goal of these CBC eviction polices is to reduce the amount of computation needed to be performed by highly constrained devices. The hope is that if a highly constrained device cannot use DTLS because of the high computational cost associated with repeatedly performing a handshake, the use of these CBC eviction policies will make it more feasible since the device would only have to connect once and then would remain connected for a long time. This would increase the number of devices using DTLS and thus, make the system more secure.

VI. Performance Evaluation

These CBC eviction policies and a few of the other traditionally used eviction policies were simulated using the CupCarbon IoT Simulator [23]. Each eviction policy was simulated with four different message transmission scenarios and with different numbers of transition nodes. The different eviction policies were then compared for each scenario to determine how the novel CBC policies compared to other traditional policies.

For each simulation there was one server node that received messages from the client nodes. The server node could maintain communication with up to 8 different client nodes before having to evict a node. This server node was programmed to evict nodes based a LRU, LFU, CO, or CSE policy. Client nodes sent messages to the server at different time intervals based on the scenario. Each message contained a unique identifier for the node and it's Level of Constraint byte. This Level of Constrain byte was assigned to each node at the beginning of each simulation.

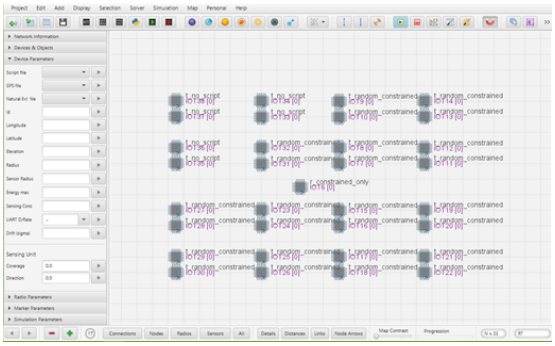

Fig. 3: Image of the simulation setup for testing in CupCarbon.

Each eviction policy was simulated with different message transmission scenarios. The first was a completely random scenario (Random). Each client node waited for a random amount of time before sending a message. They then waited for a different random amount of time before sending a new message, and so on. The second scenario was random uniform frequency (Uniform Random). At the beginning of a simulation each node was assigned a random wait time. It would send a message and then wait this length of time before sending a new message. These wait times were randomly chosen so each node could have different wait times leading to different frequencies of transmission. The next scenario involved all nodes having the same uniform waiting time which was hard-coded in at the beginning of the scenario (Uniform). The final scenario, uniform bursts created a number of items to be sent in a burst, and then took the random wait times and manipulated them to create a short and long period. The node would send out a burst of message, waiting the short period between each message, until reaching the number of items to be sent. After this, the node would then wait a long period of time before then beginning the burst of messages yet again (Random Uniform Bursts).

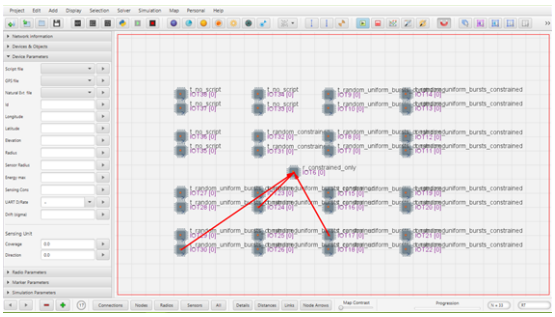

Fig. 4: Image of the simulation running in CupCarbon.

Each eviction policy and scenario was initially simulated using 8 client nodes. This is the number of nodes that can be connected so no nodes would need to be evicted. This number of nodes was increased by two for each run until reaching 32 nodes. Each simulation was run for at least 100 seconds to give each client node time to send at least 10 messages.

Each simulation recorded the Level of Constraint of each node, the number of messages sent by each node and the number of times each node needed to reconnect. The effectiveness of each eviction policy for each simulation was calculated using the Reconnect Frequency (RF). RF was calculated in using the Connections Established (CE) and Messages Sent (MS).

$$RF = \frac{\sum_{i=1}^{n}(CE_i - 1)}{\sum_{i=1}^{n}(MS_i - 1)}$$

Since the first message sent from any node will result in a connection being established, 1 was subtracted from each node in the CE and MS to look just at the reconnections.

VII. Results

Each scenario and eviction policy was coded and then run in CupCarbon. All of the code can be found in A. For items which few applications of randomness are applied, the random numbers were generated before time and then coded into the nodes. The following Reconnect Frequency were then calculated for each run and graphed vs the number of nodes per cache size.

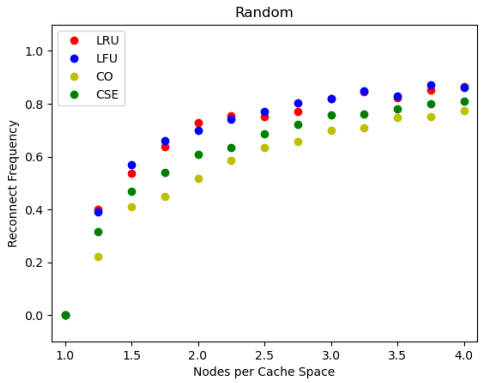

Fig. 5: Reconnect Frequency of all eviction policies where the transmitting nodes used random message transmission.

The first scenario carried out was the Random message (Fig. 5). Because there is no pattern in the receiving order of the all the time and frequency eviction policies perform poorly. The CO and CSE are able to perform better due to their full or partial lack of reliance on time and frequency. CSE splits the difference between CO and the other policies, which will be a trend that continues forward through the rest of the scenarios as well.

One scenario in which the nodes do not act the same is Uniform Random (Fig.6). The frequency of each node was

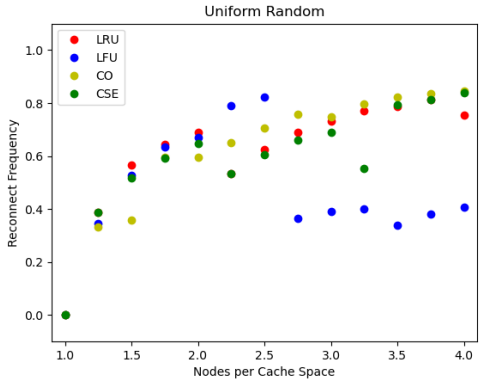

Fig. 6: Reconnect Frequency of all eviction policies where the transmitting nodes used a random uniform frequency of message transmission.

generated ahead of time, and then applied. In the RF initially the CO contains a majority of the highest frequency nodes, while the nodes on the edge of the LFU are cycled in and out of the cache, leading to more frequent reconnects. However, once a node with a high frequency which also was not heavily constrained was placed, the LFU eviction policy showed the best results by a significant margin.

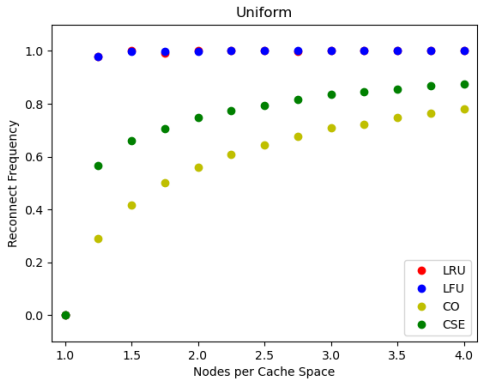

Fig. 7: Reconnect Frequency of all eviction policies where the transmitting nodes used the same uniform wait time message transmission.

As mentioned in the eviction introduction section, a pure uniform sending time will cause all connection and frequency eviction policies to fail, as seen by the nearly immediate 1.0 RF value for LRU and LFU (Fig. 7).

The final scenario considered is one in which uniform bursts are sent (Fig. 8). This scenario is where LRU is able to perform the best. This is also the scenario where CSE performs the worst.

VIII. Analysis

As mentioned above, in the case of Uniform Random messages, LFU performed worse than CO until a non-constrained

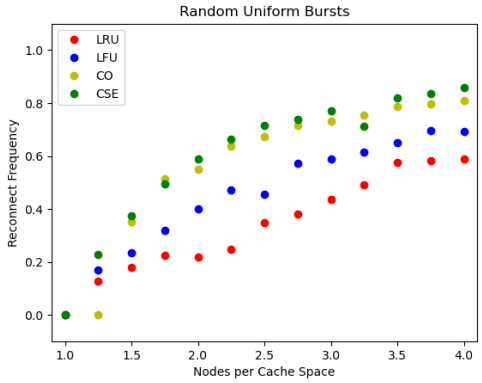

Fig. 8: Reconnect Frequency of all eviction policies where the transmitting nodes used uniform bursts message transmission.

device with a high frequency of messages was added to the system. It is important to note however, that rather than this being an outlier this is an expected outcome. Nodes which need to send a large number of messages are much more likely to have a lesser Level of Constraint because of that. If a node is operating at a high frequency, it makes much more sense for it to be plugged into a power source rather than be battery operated.

In Random Uniform Bursts, LRU performed the best due not holding onto nodes for too long after the burst ends like LFU does. Of note, this is the only location which CSE did not split the difference between CO and the others. This is due to the cache holding onto items for too long like LFU does, combined with its small LFU size only keeping a lower number of records. In this specific scenario, having a split cache with CO and LRU would be the better option over CO and LFU.

One of the largest benefits of CO and CSE are their complete consistency over the different scenarios. While they do not have the high highs of LRU and LFU, they are able to perform better in a some situations without being an extreme detriment in others. However, this may not remain true under all circumstances. In the case of high frequency but low Level of Constraint devices, a handshake would be needed for every message sent and received for CO, which could be extremely undesirable. It is also important to note that the ”high frequency” does not need to be in regards to a general sending frequency, but rather in regards to the connections being held in in the CO. If an unconstrained device is sending a message every 5 minutes, but the items in the CO send messages every 48 hours, this is a huge difference in frequency. Future testing could be conducted which looks specifically at increasing the frequency of unconstrained devices compared to the constrained ones and seeing how CO and CSE handle it.

Our proposal of the inclusion of a Level of Constraint option into CoAP is a fundamentally sound one. Since the option only takes up 2-4 bytes, the overhead incurred both during transmission and system wide is extremely small. A DTLS handshake compromises of 6 flights with up to 15 messages [12]. The client making the request will send 273 bytes of data if occurring over a perfect connection [24]. Therefor, for an extremely constrained device it would need to handshake either 69 or 137 times before the increased overhead would cause a negative impact. The entire DTLS handshake will take approximately 520 bytes, meaning that as far as network congestion is concerned, the overhead is equivalent to one handshake after either 130 or 260 times, which is nearly negligible.

The consideration of a system wide approach assumes that the byte is the only change in the system, which is likely a poor assumption to make. If the system is indeed transmitting the data, it is a reasonable assumption to make that the data is being used. As such, due to the trade off of focusing on the devices with more constraints, the overall system will likely have a need for a higher number of handshakes to occur. For IoT’s which struggle with network congestion, it is also recommended to consider not using this byte during implementation. Not only will it add to the overall amount of data being sent in the network, if a constrained cache is implemented then the overall number of handshakes occurring will also increase.

One assumption behind this solution is that the devices which are less constrained will contain significantly greater processing power and energy and these 2-4 extra bytes will be negligible. If the less constrained devices are still significantly constrained, then the extra bytes may have an unintended negative influence. If the devices are all of a similar Level of Constraint, then the byte may also be ineffective. Both of these are areas where future research could be conducted to determine the extent which the Level of Effectiveness option could have negative impacts.

One area of future work could be to use a real world networks of IoT devices as inspiration for the simulation and see how well the CO and CSE polices perform. Since, in real applications all the nodes are not going send messages in the same way as simulated in this paper. Having different nodes sending messages based on different scenarios would make the simulation better represent reality.

Three of the primary tools which IoT’s currently work off of, proxying, caching, and multicast, were not supported in this project. Multicast is currently not supported by DTLS officially, and as such does not need to be considered. Future research into how proxying and caching are able to influence CO, CSE and the Level of Constraint Option are needed, especially since proxying is able to help support devices with high Levels of Constraints.

One of the most important items to consider when reviewing the Level of Constraint option is that even if it does have a negative impact on certain networks, its inclusion as an elective, safe-to-forward option means that it can simply not be used in networks where it is sub-optimal to do so. Rather, its inclusion in CoAP ensures that the networks that are in need of it have access to it and don’t have to make

modifications to CoAP. By helping to remove the number of times that extremely constrained devices will need to perform a handshake, it means that more networks are available to utilize CoAP and DTLS, increasing their security, where they may not have been able to do so before.

IX. Conclusion

In this paper we gave a brief overview of security in IoT. We proposed a connection eviction policy based on the Level of Constraint of a given device in order to be more efficient for highly constrained devices. We have simulated and tested this and several other connection eviction policies for different node behavior scenarios.

We have shown the node behavior plays a significant role in the effectiveness in the eviction policies. Constraint Only eviction has been shown to be effective, at least at a small number of devices. CO and CSE provide the ability for more constrained devices to take part in CoAP and DTLS securely by helping reduce the number of times they will need to perform a handshake.

We suggest the addition of the Level of Constraint option into CoAP. In addition, future IoT implementations should take into account the behavior of their devices when designing their eviction policies.

References

[1] J. S. Kumar and D. R. Patel, "A survey on internet of things: Security and privacy issues," *International Journal of Computer Applications*, vol. 90, no. 11, 2014.

[2] D. M. Mendez, I. Papapanagiotou, and B. Yang, "Internet of things: Survey on security and privacy," *arXiv preprint arXiv:1707.01879*, 2017.

[3] J. Granjal, E. Monteiro, and J. S. Silva, "Security for the internet of things: A survey of existing protocols and open research issues," *IEEE Communications Surveys & Tutorials*, vol. 17, no. 3, pp. 1294–1312, 2015.

[4] T. Ramezanifarkhani and P. Teymoori, "Securing the internet of things with recursive internetwork architecture (rina)," 2018.

[5] T. Yu, V. Sekar, S. Seshan, Y. Agarwal, and C. Xu, "Handling a trillion (unfixable) flaws on a billion devices: Rethinking network security for the internet-of-things," in *Proceedings of the 14th ACM Workshop on Hot Topics in Networks*, 2015, pp. 1–7.

[6] I. S. Association *et al.*, *Ieee std 802.15. 4-2011, ieee standard for local and metropolitan area networks—part 15.4: Low-rate wireless personal area networks (lr-wpans)*, 2011.

[7] T. Winter, P. Thubert, A. Brandt, J. W. Hui, R. Kelsey, P. Levis, K. Pister, R. Struik, J.-P. Vasseur, R. K. Alexander, *et al.*, "Rpl: Ipv6 routing protocol for low-power and lossy networks.," *rfc*, vol. 6550, pp. 1–157, 2012.

[8] V. Karagiannis, P. Chatzimisios, F. Vazquez-Gallego, and J. Alonso-Zarate, "A survey on application layer protocols for the internet of things," *Transaction on IoT and Cloud computing*, vol. 3, no. 1, pp. 11–17, 2015.

[9] *The constrained application protocol (coap)*, 7252nd ed., IETF, 2014.

[10] Z. Shelby, K. Hartke, and C. Bormann, "The constrained application protocol (coap)," 2014.

[11] *Constrained restful environments (core) link format*, 6690th ed., IETF, 2012.

[12] *Datagram transport layer security version 1.2*, 6347th ed., IETF, 2012.

[13] Q. Jing, A. V. Vasilakos, J. Wan, J. Lu, and D. Qiu, "Security of the internet of things: Perspectives and challenges," *Wireless Networks*, vol. 20, no. 8, pp. 2481–2501, 2014.

[14] E. Kim, D. Kaspar, and J. Vasseur, "Design and application spaces for ipv6 over low-power wireless personal area networks (6lowpans)," *RFC6568*, 2012.

[15] S. Raza, T. Voigt, and V. Jutvik, "Lightweight ikev2: A key management solution for both the compressed ipsec and the ieee 802.15. 4 security," in *Proceedings of the IETF workshop on smart object security*, Citeseer, vol. 23, 2012.

[16] M. Malik, M. Dutta, and J. Granjal, "A survey of key bootstrapping protocols based on public key cryptog-

raphy in the internet of things," *IEEE Access*, vol. 7, pp. 27 443–27 464, 2019.

[17] S. Raza, D. Trabalza, and T. Voigt, "6lowpan compressed dtls for coap," in *2012 IEEE 8th International Conference on Distributed Computing in Sensor Systems*, IEEE, 2012, pp. 287–289.

[18] K. Hartke, "Practical issues with datagram transport layer security in constrained environments. draft-hartke-dice-practical-issues-01," *IETF work in progress*, 2013.

[19] M. Meddeb, A. Dhraief, A. Belghith, T. Monteil, and K. Drira, "How to cache in icn-based iot environments?" In *2017 IEEE/ACS 14th International Conference on Computer Systems and Applications (AICCSA)*, IEEE, 2017, pp. 1117–1124.

[20] J. Pfender, A. Valera, and W. K. Seah, "Performance comparison of caching strategies for information-centric iot," in *Proceedings of the 5th ACM Conference on Information-Centric Networking*, 2018, pp. 43–53.

[21] R. Stevens and H. Chen, "Predictive eviction: A novel policy for optimizing tls session cache performance," in *2015 IEEE Global Communications Conference (GLOBECOM)*, IEEE, 2015, pp. 1–7.

[22] C. Bormann, M. Ersue, and A. Keranen, *Terminology for constrained node networks. draft-ietf-lwig-terminology-04 (work in progress), ietf*, 2013.

[23] A. E. a. Biybceyr, "Cupcarbon: A new platform for the design, simulation and 2d/3d visualization of radio propagation and interferences in iot networks," 2018 15th IEEE Annual Consumer Communications & Networking Conference (CCNC), 2018.

[24] R. Sanchez-Iborra, J. Sánchez-Gómez, S. Pérez, P. Fernández, J. Santa, J. Hernández-Ramos, and A. Skarmeta, "Enhancing lorawan security through a lightweight and authenticated key management approach," *sensors*, 2018.

APPENDIX

Summary of the Python scripts used to program the nodes of the simulations.

- Receiver node LRU
  - Max connected nodes set to 8.
  - Create a list of connected initialized to empty.
  - When a node sends a message, reads the message.
  - If there are 8 nodes connected and this node is not currently connected, remove the first node from the list and append this node to the end of the list.Then add one count to the number of messages sent from this node and one count to the number of reconnections from this node.
  - If there are not 8 nodes connected and this node is not currently connected, append this node to end the list of connected nodes. Then add one count to the number of messages from this node and one count to the number of reconnections from this node.
  - If this node is currently connected, remove it from the list of connected nodes and then append it back onto the end of the list. Then add one count to the number of messages sent from this node

- Receiver node LFU
  - Max connected nodes set to 8.
  - Create a list of connected initialized to empty. Also create matching lists containing the number of messages sent by each node since the most recent reconnection and the time since the most recent reconnection
  - When a node sends a message, reads the message.
  - If there are 8 nodes connected and this node is not currently connected, find the node in the list of connected nodes that has sent messages least frequently since their most recent reconnection, remove it from the list of connected nodes, and append the node that just sent a message to the end of the list of connected nodes. Also update the list of number of messages sent and time since most recent reconnection. Then add one count to the number of messages sent from this node and one count to the number of reconnections from this node.
  - If there are not 8 nodes connected and this node is not currently connected, append this node to end the list of connected nodes. Then add one count to the number of messages from this node and one count to the number of reconnections from this node. Also update the list of number of messages sent and time since most recent reconnection.
  - If this node is currently connected, add one to the number of messages sent since last reconnection. Then add one count to the number of messages sent from this node.

- Receiver node CO
  - Max connected nodes set to 8.
  - Create a list of connected initialized to empty.
  - When a node sends a message, reads the message.
  - If there are 8 nodes connected and this node is not currently connected, remove the first node from the list and append this node to the end of the list.Then add one count to the number of messages sent from this node and one count to the number of reconnections from this node.
  - If there are not 8 nodes connected and this node is not currently connected, append this node to end the list of connected nodes. Then add one count to the number of messages from this node and one count to the number of reconnections from this node.
  - If this node is currently connected, add one count to the number of messages sent from this node
  - Next sort the list of connected nodes by Level of Constraint that is sent in the message from the node so that the least constrained node is at the beginning of the list and will be the first to be removed.

- Receiver node CSE
  - Max connected nodes set to 4.
  - Create two lists, Constrained and Frequency, of connected initialized to empty.
  - When a node sends a message, reads the message.
  - If the Constrained isn't full and this node isn't connected, append this node to the list. Then add one count to the number of messages from this node and one count to the number of connections.
  - If the Frequency isn't full and this node isn't connected, append this node to the list, including the time it connected and an current received value of one. Then add one count to the number of messages from this node and one count to the number of connections.
  - If this node is currently connected to the Constrained list, add one count to the number of messages received from this node.
  - If this node is currently connected to the Frequency list, add one count to the current received value and one count to the number of messages from this node.
  - If the node has a higher Level of Constraint than the least constrained node on the Constrained list, remove the least constrained node and add this node. Then add one count to the number of messages from this node, and one count to the number of connections.
  - Sort the Frequency list to determine the least frequent item, remove it, and add this node. Then add one count to the number of messages from this node, and one count to the number of connections.

- Transmitter node Random
  - From the preset list of constraint values select the one matching this node's id value.
  - Randomly generate a wait period.
  - Wait that length of time then send a message containing the name of this node and it's Level of Constraint
  - Repeat until simulation ends

- Transmitter node Uniform Random
  - From the preset list of constraint values and wait times select the ones matching this node's id value.
  - Wait that length of time then send a message containing the name of this node and it's Level of Constraint
  - Repeat until simulation ends

- Transmitter node Uniform
  - From the preset list of constraint values select the one matching this node's id value.
  - Randomly generate a wait period.
  - Wait that length of time then send a message containing the name of this node and it's Level of Constraint.
  - Wait for 5.0 seconds then send the same message as before.
  - Continue waiting 5.0 seconds between sending each message until the simulation ends.

- Transmitter node Random Uniform Bursts
  - From the preset list of constraint values and wait times select the ones matching this node's id value.
  - Multiply the wait time by 3 to get the long wait period.
  - Multiply the wait time by 0.5 to get the short wait period.
  - Round the wait time to the nearest integer to get the number of messages per burst.
  - Wait the short time then send a message containing the name of this node and it's Level of Constraint.
  - Repeat waiting for the short time then sending messages until sending a number of messages equal to the number of messages per burst found earlier.
  - Wait for the long wait time.
  - Repeat until simulation ends