Power Consumption Analysis of the New Covert Channels in CoAP

Aleksandar Velinov, Aleksandra Mileva, Done Stojanov
Faculty of Computer Science
University “Goce Delčev”
Štip, Republic of Macedonia
email: {aleksandar.velinov, aleksandra.mileva, done.stojanov}@ugd.edu.mk

***Abstract* — This paper presents several novel covert channels in the Constrained Application Protocol (CoAP) - a specialized Web transfer protocol used for Internet of Things. The suggested new covert channels are categorized according to the pattern-based classification, and, for each covert channel, the total number of hidden data bits transmitted per second, or per protocol data unit is given, together with the theoretical performance evaluation. Additionally, we present a methodology for power consumption analysis of these covert channels, and we give the experimental results of applying this methodology for one of the discovered CoAP covert channels.**

Keywords - CoAP; network steganography; Contiki OS; Cooja; Copper.

I. INTRODUCTION

We have investigated several novel covert channels for the Constrained Application Protocol (CoAP) in the conference paper [1]. In this paper, we extend the results with an experimental methodology for power consumption analysis of these covert channels, and we give the experimental results of applying this methodology for one of the discovered CoAP covert channels.

Network steganography is a practice of hiding data in legitimate transmissions in communication networks, by deploying different network protocols as carriers, and concealing the presence of hidden data from network devices. It provides only security through obscurity.

Covert channels are first introduced by Lampson [10] as channels “not intended for information transfer at all” and they can be exploited by a process to transfer information in a manner that violates the systems security policy. The current distinction between the network steganography and covert channels is artificial, especially in a communication networks environment. Network steganography techniques create covert channels for hidden communication, but such covert channels do not exist in communication networks without steganography [14]. There is no some algorithm for exaustive search of all covert channels in a given protocol.

Covert channels can be divided in two basic groups: storage and timing channels. Storage covert channels are channels where one process writes (directly or indirectly) to a shared resource, while another process reads from it. In the context of network steganography, storage covert channels hide data by storing them in the protocol header and/or in the Protocol Data Unit (PDU). On the other hand, timing channels hide data by deploying some form of timing of events, such as retransmitting the same PDU several times, or changing the packet order.

Network-based covert channels may have black hat or white hat applications. Black hat applications include coordination of distributed denial of service attacks, spreading of malware (for example, by hiding command and control traffic of botnets), industrial espionage, secret communication between terrorists and criminals, etc. On the other hand, white hat applications include covert military communication in hostile environments, prevention of detection of illicit information transferred by journalists or whistle-blowers, circumvention of the limitation in using Internet in some countries (e.g., Infranet [4]), providing Quality of Service - QoS for Voice over Internet Protocol - VoIP traffic [12], secure network management communication [6], watermarking of network flows (e.g., RAINBOW [8]), tracing encrypted attack traffic or tracking anonymous peer-to-peer VoIP calls [21][22], etc.

Nowadays, there are a plenty of choices in the landscape of network protocols for carriers. There are several surveys about different covert channels in many TCP/IP (Transmission Control Protocol/Internet Protocol) protocols [15][26]. To the best of our knowledge, there are only a few papers about network steganographic research addressing protocols specialized for constrained devices in the IoT (sensors, vehicles, home appliances, wearable devices, and so on) [3] [9]. The Constrained Application Protocol (CoAP) [19] is a specialized Web transfer application layer protocol, which can be used with constrained nodes and constrained networks in the IoT. The nodes are constrained because they have 8-bit microcontrollers, for example, with limited random-access memory (RAM) and read-only memory (ROM). Constrained networks often have high packet error rates and small data rate (such as IPv6 over Low-Power Wireless Personal Area Networks - 6LoWPANs). CoAP is designed for machine-to-machine (M2M) applications and its last stable version was published in June 2014 in the RFC 7252 [19]. In fact, it is a Representational State Transfer - RESTful protocol with multicast and observe support. In this paper, we try to apply existing network steganographic techniques for creating covert channels in CoAP.

Wendzel et al. [24] presented a new pattern-based categorization of network covert channel techniques into 11 different patterns or classes. They represented the patterns in a hierarchical catalog using the pattern language Pattern

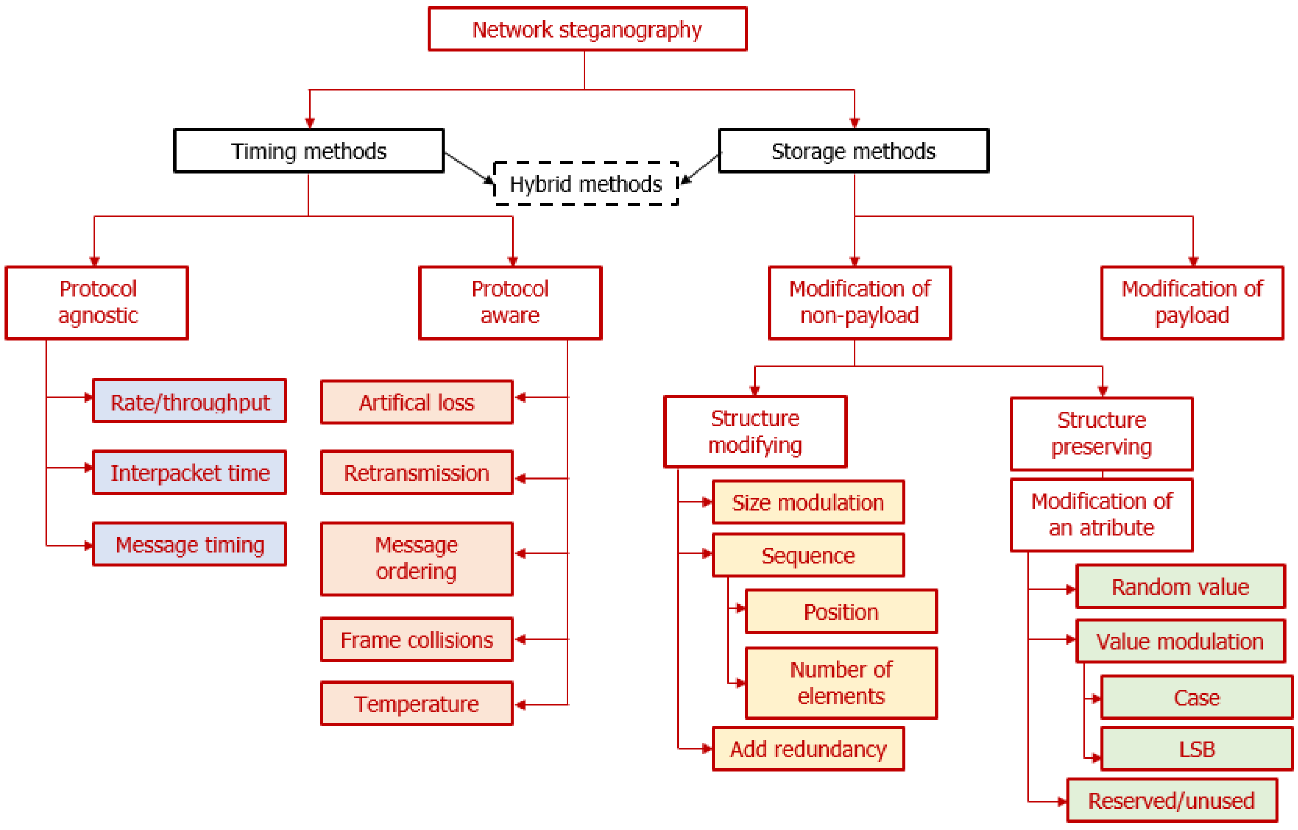

Figure 1. Pattern-based categorization of network covert channel techniques

Language Markup Language (PLML) v. 1.1 [5]. A modification of this categorization is made by Mazurczyk et al. [14]. In our paper, we use this joint classification (see Figure 1) to characterize our covert channels.

Covert channels are analyzed through the total number of hidden data bits transmitted per second (Raw Bit Rate - RBR), or through the total number of hidden data bits transmitted per PDU (for example, Packet Raw Bit Rate-PRBR) [13]. For each new CoAP channel, its PRBR value is given, where PDU is a CoAP message.

The rest of this article is structured as follows. The related work is presented in Section II. Details about the CoAP header, messages, functionalities and concepts are presented in Section III. The main Section IV describes eight groups of new covert storage and timing channels in CoAP, that can be used regardless its transport carrier (DTLS or clear UDP). Some possible applications of these covert channels are also briefly suggested in this section. In Section V we present the performance evaluation, while the experimental evaluation of one of the new covert channels is given in Section VI. We conclude the paper in Section VII.

II. Related Work

The research on network steganography for IoT has seen an increased interest recently.

One example for this is the work of Islam et al. [9], which uses Internet Control Message Protocol (ICMP) covert channels for authenticating Internet packet routers as an intermediate step towards proximal geolocation of IoT devices. This is useful as a defense from the knowledgeable adversary that might attempt to evade or forge the geolocation. Hidden data are stored in the data field of the ICMP Echo Request and ICMP Echo Reply messages.

Patuck et al. [18] present several storage covert channels in the Extensible Messaging and Presence Protocol (XMPP), a popular instant messaging protocol based on XML, which in the past was used by many messaging platforms such as Google Talk, AOL Instant Messenger, Microsoft Messenger Sevice, etc. These covert channels use some attributes in the XMPP messages, like Type, id and xml:lang attributes, or the message body. For example, for the Type attribute, three covert channels are presented: by changing cases of the value, by changing value, or by presence/absence of the attribute.

A storage covert channel with modulated sensor readings is presented by Tuptuh et al. [20] for wireless sensor networks. In this channel, LSBs of encrypted sensor readings are the cover bits. The sender performs the following algorithm: while LSB bit of the current reading is different from the cover bit, small offset is added to the sensor reading (e.g., temperature) and the value is encrypted.

Building Automation and Control Networking Protocol (BACnet) is another protocol for which two storage (message-type based and parameter-based) and one timing (with inter packet gaps) covert channels were given by Wendzel [23].

Wendzel et al. [25] have showed that even a cyber-physical system (CPS) can be used for network steganography. One can places hidden data in the CPS environment by slightly modifying some of its components, like actuators, sensors, controllers, and monitoring equipment. The authors apply the term scatter hoarding, which means that only small modifications of the CPS will be allowed but they will be applied to numerous carefully selected components, to avoid them being regularly modified, e.g., by a human user. One example is the temperature sensor, which comprises two 8-bit alarm registers with a lower and an upper warning threshold, and can be used to store hidden values. Another example is the state modulation of actuators, like heater, in which the heating value of 80% will be a binary “0”, and of 79% will be a binary “1”. Because the actuator states change and influence the physical environment, steganographic operations may not be robust and be easily detectable and thus need a reasonable storage strategy.

Some applications of steganography in IoT are not connected with the protocols themselves, but with the applications on top of these protocols. For example, Denney et al. [3] present a novel storage covert channel on wearable devices that sends data to other applications, or even to other nearby devices, through the use of notifications that are normally displayed on the status bar of an Android device. For that purpose, a notification listening service on the wearables needs to be implemented. Data are hidden in the notification ID numbers (32 bits), and their exchange is done by using two functions notify and cancel. If the notifying function is immediately followed by the canceling function, the notification is never displayed to the user although it can be seen in the log files, so the communication is hidden from the user who wears the device.

There are several papers that deploy steganography in the physical or medium access control (MAC) layers [7][11][16][19].

As far as we now, there is no paper (other than [1]) that analyze existance of covert channels in CoAP. Additionally, we try to give a methodology how one can perform a power consumption analysis of a given covert channel in the IoT device.

III. How CoAP Works

Similar to HTTP, CoAP uses client/server model with request/response messages. It supports built-in discovery of services and resources, Uniform resource identifiers (URIs) and Internet media types. The CoAP sends request message requesting an action (using a Method Code) to the resource (idenified by a URI) hosted on server. The server responds to this request by using the response message that contains the Response Code, and possibly some resource representation. CoAP defines four types of messages: Confirmable (CON), Non-Confirmable (NON), Acknowledgment (ACK) and Reset (RST). These types of messages use method and response codes to transmit requests or answers. The requests can be transmitted as Confirmable and Non-Confirmable types of messages, while the responses can be transmitted through these and via piggybacked and Acknowledgment types of messages.

CoAP uses clear UDP or DTLS on transport layer to exchange messages asynchronously between endpoints. As shown in Figure 2, each message contains a Message ID used for optimal reliability and to detect duplicates. A message that requires reliable transmission is marked as CON, and if does not, it is marked as NON. The CON message is retransmitted using a default timeout and binary exponential back-off algorithm for increasing the timeout between retransmissions, until the recipient sends an ACK message with the same Message ID. When the recipient is not able at all to process CON or NON message, it replies with a RST message.

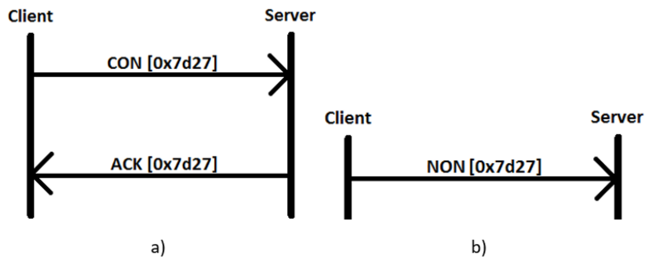

Figure 2. a) Reliable CoAP message transmission b) Unreliable CoAP message transmission.

CoAP messages are encoded into simple binary format (see Figure 3). Each message starts with a 4B fixed header, followed by a Token field, with size from 0 to 8B. Then comes the optional Options field and optional Payload field. If the Payload field is present it is preceded by one-byte Payload Marker (0xFF).

The fields that make up the message header are the following:

- Version (Ver) - 2-bit unsigned integer that idenitfies the CoAP version. Currently it must be set to 01.
- Type (T) – 2-bit unsigned integer that indicates the message type: Confirmable (0), Non-Confirmable(1), Acknowledgement (2), or Reset (3).
- Token Length (TKL) – 4-bit unsigned integer that stands for the length of the Token field (0-64 bits). Lengths 9-15 are reserved and must be processed as a message format error.
- Code – 8-bit unsigned integer. It is divided into two parts: 3-bit class (the most significant bits) and 5-bit details (the least significant bits). The format of the code is “c.dd”, where “c” is a digit from 0 to 7 and represents the class while “dd” are two digits from 00 to 31. According to the class we can determine the type of the message, such as: request (0), a successful response (2), a client error response (4), or a server error response (5).

CoAP has a separate code registry that provides a description for all codes [2].

- Message ID - 16-bit unsigned integer that is used to detect duplicate messages and to connect Acknowledgment/Reset messages with Confirmable/ Non-Confirmable messages.

The message header is followed by the Token field with variable size from 0 to 64 bits. This field is used to link requests and responses.

The optional Options field defines one or more options. CoAP defines a single set of options that are used both for requests and for responses. These are: Content-Format, Etag, Location-Path, Location-Query, Max-Age, Proxy-Uri, Proxy-Scheme, Uri-Host, Uri-Path, Uri-Port, Uri-Query, Accept, If-Match, If-None-Match, and Size1.

The payload of requests/responses that indicates success typically carries the resource representation or the result of the requested action.

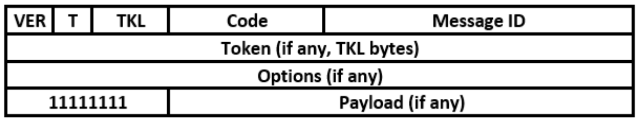

Figure 3. CoAP message format.

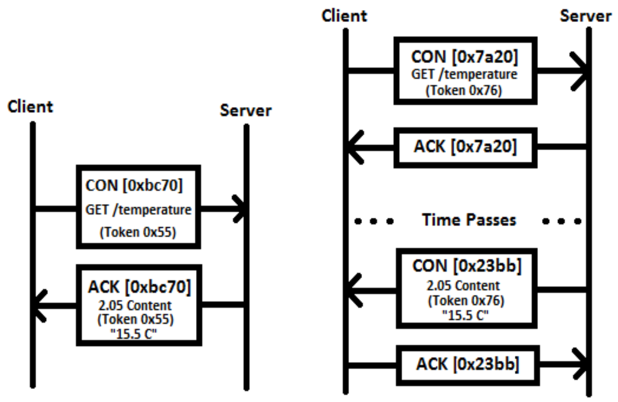

Figure 4. a) Piggybacked response b) Separate response.

There are two types of responses: piggybacked and separate (Figure 4). If the request is transmitted via CON or NON message, and if the response is available and transmitted via an ACK message, then it is piggybacked response. If the server is unable to respond immediately to the request, an Empty message (with code 0.00) is sent that tells the client to stop sending the request. If the server is able for later respond to the client, it sends a CON message that must then be confirmed by the client. This is called a separate response.

Similar to HTTP, CoAP uses GET (with code 0.01), POST (with code 0.02), PUT (with code 0.03), and DELETE (with code 0.04) methods.

IV. New Covert Channels in the CoAP

When someone creates a covert channel (CC) in network protocol, usually uses: a protocol feature that has a dual nature (i.e., the same feature can be obtained in more than one way), a feature that is not mandatory, a feature that can obtain random value, and so on. Therefore if we use some of these features, we can create new covert channels in CoAP. From the beginning, CoAP offers some protection against network steganography. For example, by introducing a proper order in the appearance of different options in message, the steganographic techniques that deploy different order of options can not be applied.

CoAP can be applied in different fields, such as: smart energy, smart grid, building control, intelligent lighting control, industrial control systems, asset tracking, environment monitoring, and so on. So, one useful scenario of application of the CoAP covert channels would be for support of the authentication of geolocation of IoT devices. Another possible scenario is clandestine communication between wearable devices in a hostile environment, for the needs of the soldiers, or, between nodes in a wireless sensor network.

As steganography offers only security through obscurity, a successful attack against any covert channel consists in detecting the existence of this communication. Next, the new CoAP covert channels are presented.

A. *Covert Channel Using Token and/or Message ID Fields*

The Message ID contains a random 16-bit value. In the case of piggybacked response for CON message, the Message ID should be the same as in the request, while in the case of separate response, the server generate different random Message ID (while the request Message ID is copied in the first sent Empty ACK message).

The same Message ID can not be reused (in the communication between same two endpoints) within the EXCHANGE_LIFETIME, which is around 247 seconds with the default transmission parameters.

The Token is another random generated field, with variable size up to 64 bits, used as a client-local identifier to make a difference between concurrent requests. If the request results in the response, the Token value should be echoed in that response. This also happens in the case when the server sends separate response. So, we can create an unidirectional or a bidirectional communication channel between two hosts, by sending 16 (from Message ID) plus/or 64 (from Token ID) bits per message (PRBR $\in \{16, 64, 80\}$). According to the pattern-based classification [14][24], this channel belongs to the following class:

```
Network Covert Storage Channels
 --Modification of Non-Payload
  --Structure Preserving
    --Modification of an Attribute
     --Random Value Pattern
```

B. *Covert Channel Using Piggybacked and Separate Response*

Since the server has a choice for sending piggybacked or separate response, one can create an one-bit per message unidirectional or a bidirectional covert channel (PRBR=1), such as:

- piggybacked response to be binary 1, and
- separate response to be binary 0.

At heavy load, the server may not be able to respond (sending binary 1), so this covert channel is limited to the times when the server has the choice. According to the pattern-based classification [14][24], this channel belongs to the following class:

```
Network Covert Timing Channels
 --Protocol aware
  --Message ordering pattern
```

C. *Covert Channel Using Payload of the Message*

Both requests and responses may include a payload, depending of the Method or the Response Code, respectively. Its format is specified by the Internet media type and content coding providen by the Content-Format option. The payload of requests or of responses that indicates success is typically a representation of the resource or the result of the requested action.

If no Content-Format option is given, the payload of responses indicating client or server error is a Diagnostic Payload, with brief human-readable diagnostic message being encoded using UTF-8 (Unicode Transformation Format) in Net-Unicode form.

The CoAP specification provides only an upper bound to the message size - to fit within a single IP datagram (and into one UDP payload). The maximal size of the IPv4 datagram is 65,535B, but this can not be applied to constrained devices and networks. According to IPv4 specification in the RFC 791, all hosts have to be prepared to accept datagrams of up to 576B, while IPv6 requires the maximum transmission unit (MTU) to be at least 1280B. The absolute minimum value of the IP MTU for IPv4 is 68B, which would leave at most 35B for a CoAP payload (the smallest CoAP header size with Payload Marker before the payload is 5B, assuming 0B for Token and no options). On the other hand, constrained network presents another restriction. For example, the IEEE 802.15.4's standard packet size is 127B (with 25B of maximum frame overhead), which leaves (without any security features) 102B for upper layers. The sizes of the input/output buffers in the constrained devices are another restriction of the maximal payload. Thus, we can create a unidirectional or a bidirectional communication channel between two hosts, by sending a Diagnostic Payload with the smallest maximal size of 35B per message (PRBR=280). According to the pattern-based classification [14][24], this channel belongs to the following class:

```
Network Covert Storage Channels
 --Modification of Payload Pattern
```

Another similar channel can be created by encoding the data in some specific Internet media format (for example, “application/xml” media type) and sending this format as payload of a message with appropriate Content-Format option (41 for “application/xml”).

D. *Covert Channel Using Case-insensitive Parts of the URIs*

CoAP uses “coap” and “coaps” URI (Uniform Resource Identifier) schemes for identification of CoAP resources and providing a means for locating the resource. The URIs in the request are transported in several options: URI-host, URI-Path, URI-Port and URI-Query. They are used to specify the target resource of a request to CoAP origin server. The URI-host and the scheme are case insensitive, while all other components are case-sensitive. So, we can create a unidirectional covert channel between the client and the server using, for example:

- capital letter in the URI-host option to be binary 1, and
- small letter in the URI-host option to be binary 0.

Taking into account that valid Domain Name System (DNS) name has at most 255B, we can send at most 255B per message, or in other words, the PRBR of this channel is up to 255B. According to the pattern-based classification [14][24], this channel belongs to the following class:

```
Network Covert Storage Channels
 --Modification of Non-Payload
  --Structure Preserving
   --Modification of an Attribute
    --Value Modulation
     --Case Pattern
```

CoAP supports proxying, where proxy is a CoAP endpoint that can be tasked by CoAP clients to perform requests on their behalf. Proxies can be explicitly selected by clients, using the Proxi-URI option, and this role is “forward-proxy”. Proxies can also be inserted to stand in for origin servers, a role that is named as "reverse-proxy". So, we can create similar covert channel using schema and host part from the Proxi-URI option. A request containing the Proxy-URI Option must not include URI-host, URI-Path, URI-Port and URI-Query options.

E. *Covert Channel Using PUT and DELETE Methods*

The PUT method requires the resource identified by the URI in the request, to be updated or created with the enclosed representation. If the resource exists at the request URI, the enclosed representation should be considered as a modified version of that resource, and a 2.04 (Changed) Response Code should be returned. If no resource exists, then the server may create a new resource with the same URI that results in a 2.01 (Created) Response Code.

The DELETE method requires deletion of the resource, which is identified by the URI in the request. Regardless if the deletion is successful, or the resource did not exist before the request, a 2.02 (Deleted) Response Code should be send.

If somebody has a known representation of the existing resource R1 on the server and if he knows that specific resource R2 does not exist on the same server, a unidirectional covert channel to the server can be created, in this way:

- send request with PUT method to create the resource R1 with enclosed known representation as binary 1, and
- send request with DELETE method to delete non-existing resource R2 as binary 0.

In this way, one bit per message can be sent (PRBP=1). According to the pattern-based classification [14][24], this channel belongs to the following class:

```
Network Covert Storage Channels
 --Modification of Non-Payload
  --Structure Preserving
   --Modification of an Attribute
    --Value Modulation Pattern
```

F. Covert Channel Using Accept Option

The Accept option can be used to indicate which Content-Format is acceptable to the client. If no Accept option is given, the client does not express a preference. If the preferred Content-Format if available, the server returns in that format, otherwise, a 4.06 "Not Acceptable" must be sent as a response, unless another error code takes precedence for this response. We can create a unidirectional one-bit per message covert channel (PRBP=1), in this way:

- sending a given message without Accept option to be binary 1, and
- sending a given message with Accept option to be binary 0.

According to the pattern-based classification [14][24], this channel belongs to the following class:

```
Network Covert Storage Channels
 --Modification of Non-Payload
  --Structure Modifying
   --Add Redundancy Pattern
```

G. Covert Channel Using Conditional Requests

Conditional request options If-Match and If-None-Match enable a client to ask the server to perform the request only if certain conditions specified by the option are fulfilled. In the case of multiple If-Match options the client can make a conditional request on the current existence or value of an ETag for one or more representations of the target resource. This is useful to update the request of the resource, as a means for protecting against accidental overwrites when multiple clients are acting in parallel on the same resource. The condition is not fulfilled if none of the options match. With If-None-Match option the client can make a conditional request on the current nonexistence of a given resource. If the target resource does exist, then the condition is not fulfilled.

If somebody knows for sure that given condition C1 is fulfilled (for example, the resource is created or deleted in previous message) and other C2 is not fulfilled, using either of If-Match and If-None-Match options, a unidirectional one-bit per message covert channel (PRBP=1) can be created in this way:

- sending a given message without fulfilled condition to be binary 1 (e.g., If-Match + C2), and
- sending a given message with fulfilled condition (e.g., If-Match + C1) to be binary 0.

According to the pattern-based classification [14][24], this channel belongs to the following class:

```
Network Covert Storage Channels
 --Modification of Non-Payload
  --Structure Preserving
   --Modification of an Attribute
    --Value Modulation Pattern
```

H. Covert Channel Using Re-Transmissions

If we are using CoAP in channels with small error-rate (to cope with the unreliable nature of UDP), we can create a unidirectional or a bidirectional covert channel using retransmissions with PRBP=1, in this way:

- sending a given message only once to be binary 1, and
- sending a given message two or more times to be binary 0.

In this way, one bit per message can be sent. According to the pattern-based classification [14][24], this channel belongs to the following class:

```
Network Covert Timing Channels
--Protocol aware
  --Re-Transmission pattern
```

V. Performance Evaluation

Suppose that two IoT devices communicate with CoAP every t seconds.

Any covert channel with a given PRBR will need at least

$$\mathrm{ceil}(l \,/\, \mathrm{PRBR}) \cdot t\ (s)$$

for sending a message with length l bits.

We can evaluate the minimum time for sending the message ”Hello, world!” using the newly suggested covert channels. The message has length of 13 7-bit ASCII characters or l=91 bits. Results are given in Table I.

So, we can see that not all suggested covert channels in CoAP are able to send short messages in real time, especially the ones with PRBR=1. Still, the covert channels 3 and 4 can be used for sending a short message per one CoAP message, without rising any suspicions. If the time for sending the message is not so important, one can choose covert channels 1 or 2, without rising any suspicions.

TABLE I. PERFORMANCE EVALUATION OF THE NEW COVERT CHANNELS FOR SENDING THE MESSAGE "HELLO, WORLD!"

No.	Type of CC	PRBR	Time (*s*)		
			t=1*s*	*t*=5*s*	*t*=10*s*
1	CC using token and/or message ID Fields	16	6	30	60
		64	2	10	20
		80	2	10	20
2	CC using piggybacked and separate response	1	91	455	910
3	CC using payload of the message	280	1	1	1
4	CC using case-insensitive parts of the URIs	≤2040	1	1	1
5	CC using PUT and DELETE Methods	1	91	455	910
6	CC using Accept option	1	91	455	910
7	CC using conditional requests	1	91	455	910
8	CC using re-transmissions	1	91	455	910

TABLE II. PERFORMANCE EVALUATION OF THE NEW COVERT CHANNELS WITH PRBR>1 FOR SENDING 320X240 RAW COLOR IMAGE (WITH 24-BIT PIXELS)

	Type of CC	PRBR	Time(*s*)	
			t=1*s*	*t*=5*s*
1	CC using token and/or message ID Fields	16	115200 (32h)	576000 (160h)
		64	28800 (8h)	144000 (40h)
		80	23040 (6,4h)	115200 (32h)
2	CC using payload of the message	280	6583 (>1,82h)	32915 (>9.1h)
3	CC using case-insensitive parts of the URIs	≤2040	904 (15 min)	4520 (76 min)

Additionally, we can evaluate the minimum time for sending the 320x240 raw color image (with 24-bit pixels) using the newly suggested covert channels. The size of the image is 225KB or *l*=1843200 bits. Results are given in Table II.

The results from Table II show that most of the new CoAP covert channels are not quite suitable for sending images, because of the large transmission time. The covert channel 3 is the most suitable for that purpose (it will send 225KB image in 15 minutes).

VI. EXPERIMENTAL EVALUATION

For our research we have used Contiki OS, and specially, Instant Contiki version 3.0 as a development environment. It is a Ubuntu Linux virtual machine that runs in VMWare player. It has all the development tools, compilers and simulators. We can develop our application and test it on one of the devices in simulator. We used Cooja simulator. With it, we can create different types of devices for which we can develop applications. This is practical because before we execute our application on real device we will make sure it works properly.

For the purposes of our research we used Z1 Zolertia Mote. It is an ultra low power wireless module for use in wireless sensor networks (WSN). Z1 has the second generation of MSP430F2617 low power microcontroller, which has a powerful 16-bit RISC CPU @16MHz clock speed. It also has built-in clock factory calibration, 8KB RAM and a 92KB Flash memory. Z1 module includes the CC2420 transceiver, which operates at 2.4GHz with data rate of 250Kbps and it supports 802.15.4 standard to interoperate with other devices. This module has a built-in temperature and 3-axis accelerometer sensors. Z1 allows flexible powering using the battery pack (2xAA or 2xAAA), Coin Cell, USB and with direct connection.

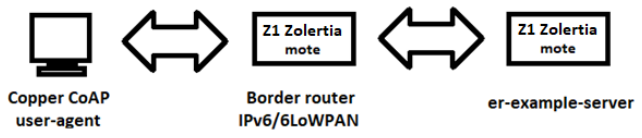

Figure 5. Implementation scenario

In our research, we used Copper (Cu) as a CoAP user-agent. It is a Firefox plugin that installs a handler for "coap" URI scheme and allows users to browse and interact with Internet of Things (IoT) devices. The scenario for our research is presented on Figure 5. We used the Cooja simulator to create a new simulation with, 2 Z1 Zolertia motes. One Z1 mote is for Border Router. As a source we used rpl-border router source code that is located in:

/home/user/contiki-3.0/examples/ipv6/rpl-border-router

The other Z1 mote is for CoAP server. In our research we used Erbium implementation of CoAP server for Contiki OS. The source code is located in:

/home/user/contiki-3.0/examples/er-rest-example

To adjust it to our needs, we made a change to the source code, specifically in the file “res-hello.c” that is located in the following path in Contiki:

/home/user/contiki-3.0/examples/er-rest-example/resources

According to this scenario, we have implemented the covert channel that uses the PUT and DELETE methods. By using Copper user-agent, we created request using the PUT and DELETE methods (with PUT in the 50th second of the execution of the simulation and with DELETE in the 60th second of the execution of the simulation). We also examined power consumption in case when we do not implement a covert channel and in the case of an implemented covert channel. To calculate the power consumption, we used the data obtained with the tool “Powertrace” for CoAP server with and without implemented covert channel. These data are printed at the mote output for Z1 module in Cooja. We made the calculations in a total time interval of 100 seconds as previously predefined interval for performing the simulation for both cases. These data show the total number of clock ticks in different states of the module: CPU (CPU in active mode), LPM (CPU in Low Power Mode), TX (Transmit) and RX (Receive) (Table III and Table IV).

TABLE III. DATA OBTAINED WITH “POWERTRACE” FOR “COAP” SERVER WITHOUT IMPLEMENTATION OF COVERT CHANNEL

ALL_CPU	ALL_LPM	ALL_TX	ALL_RX
4674	322863	149	294987
9879	645197	229	622586
15204	967576	412	950244
17500	1292676	412	1277763
19778	1617956	412	1605442
24933	1940346	514	1933022
27435	2265399	594	2260619
29721	2590672	594	2588298
32001	2915952	594	2915978
34271	3241241	594	3243658
39491	3563562	675	3571258

To calculate the power consumption, we used the following formula [27] :

$$Power\_consumptio = \frac{Energest\_value * Current * Voltage}{RTIMER\_SECOND * Runtime}$$

Energest_value is the difference between the number of clock ticks (in states CPU, LPM, TX and RX) between two time intervals. We used the Z1 datasheet to get the values for *Current* in different states (Approximate Current Consumption of Z1 circuits: Active Mode @16MHz - < 10 mA (approximate 9mA), Standby Mode - 0.5µA, RX Mode - 18.8mA, TX Mode - 17.4mA) [28]. The value for *Voltage* parameter is 3V. The value for *RTIMER_SECOND* is 32768. *Runtime* is the time interval (10 seconds in our case).

TABLE IV. DATA OBTAINED WITH “POWERTRACE” FOR “COAP” SERVER WITH IMPLEMENTED COVERT CHANNEL

ALL_CPU	ALL_LPM	ALL_TX	ALL_RX
4726	322829	149	294987
10020	645086	229	622586
15271	967393	332	950165
17738	1292480	413	1277762
20253	1617524	476	1605379
25731	1939607	641	1932896
28236	2264657	722	2260493
30521	2589930	722	2588173
32801	2915210	722	2915853
35071	3240499	722	3243533
40372	3562751	802	3571132

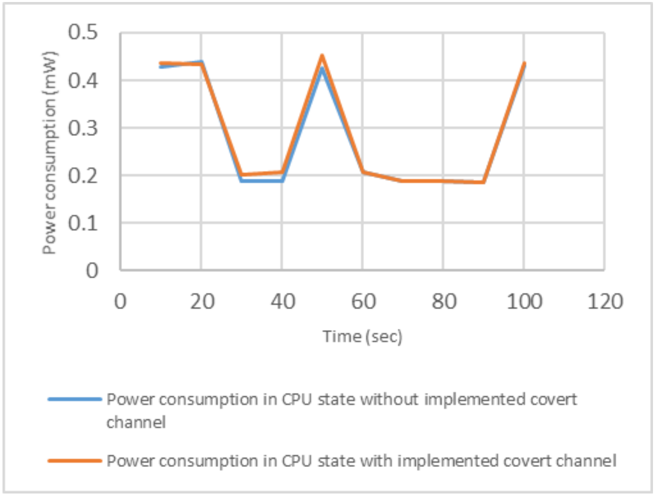

Figure 6. Power consumption for CoAP server (Z1) in CPU state (with and without implemented covert channel)

Figure 6 shows the power consumption for Z1 module (implemented as CoAP server) in CPU state with and without implemented covert channel. The average power consumption without implemented covert channel is 0.28688324 mW, while the average power consumption with implemented covert channel is 0.293713989 mW. We can see that the average power consumption with an implemented covert channel is bigger.

Figure 7 shows the power consumption for Z1 module (implemented as CoAP server) in LPM state with and without implemented covert channel. The average power consumption without implemented covert channel is 0.001483474 mW, while the average power consumption with implemented covert channel is 0.001483119 mW. We can see that the average power consumption with implemented covert channel is slightly smaller than the power consumption without implemented covert channel.

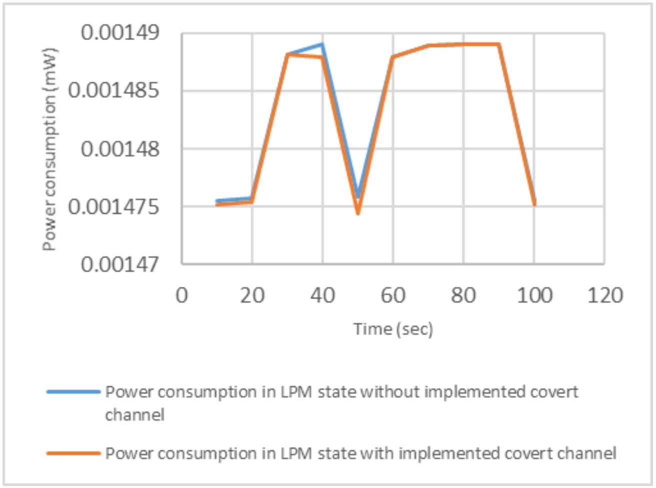

Figure 7. Power consumption for CoAP server (Z1) in LPM state (with and without implemented covert channel)

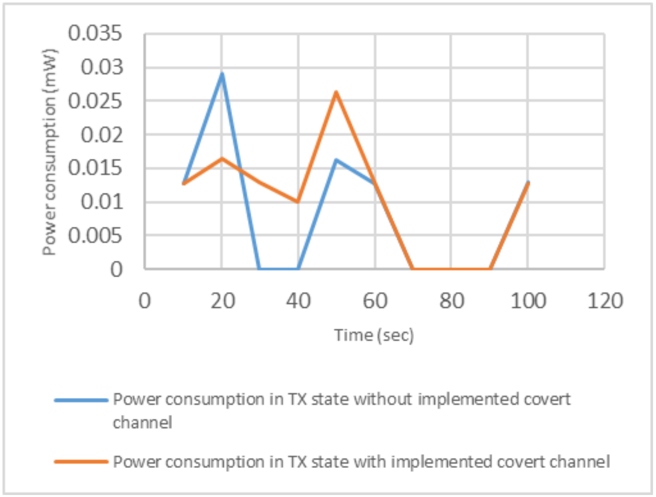

Figure 8. Power consumption for CoAP server (Z1) in TX state (with and without implemented covert channel)

Figure 8 shows the power consumption for Z1 module (implemented as CoAP server) in TX state with and without implemented covert channel. The average power consumption without implemented covert channel is 0.008379272 mW, while the average power consumption with implemented covert channel is 0.010402405 mW. We can see that the power consumption with implemented covert channel is around 1.24 times greater than the power consumption without implemented covert channel.

Figure 9 shows the power consumption for Z1 module (implemented as CoAP server) in RX state with and without implemented covert channel. The average power consumption without implemented covert channel is 56.3908949 mW, while the average power consumption with implemented covert channel is 56.3887262 mW.

We can see that the average power consumption with implemented covert channel is slightly smaller than the power consumption without implemented covert channel.

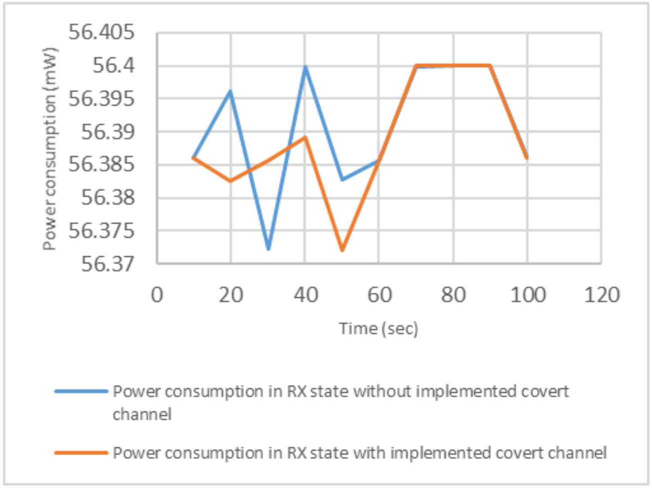

Figure 9. Power consumption for CoAP server (Z1) in RX state (with and without implemented covert channel)

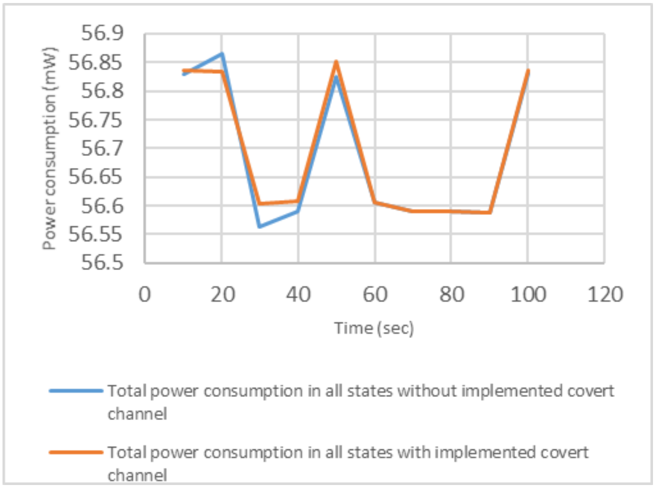

Figure 10. Total power consumption for CoAP server (Z1) in all states (with and without implemented covert channel)

Figure 10 shows the total power consumption for Z1 module (implemented as CoAP server) in all states, in each time interval with and without implemented covert channel. The average power consumption without implemented covert channel is 56.68764088mW, while the average power consumption with implemented covert channel is 56.69432571mW. We can see that the average power consumption (in all states) for Z1 module with implemented covert channel (when sending two bits) is slightly greater than the power consumption without implemented covert channel.

The power consumption of the Z1 module in the 50th second (the time when we sent a request with the PUT method) with implemented covert channel has increased very little, for only 0.02580548 mW.

We have the same case in the 60th second (the time when we sent a request with the DELETE method), when the power consumption with implemented covert channel has increased very little, for only 0.00040648 mW. The implementation of the covert channel using the PUT and DELETE methods does not greatly affect the power consumption of the Z1 module.

VII. CONCLUSION

Considering that IoT will consist of about 30 billion objects by 2020 [17], CoAP belongs to the group of possible most exploited protocols in the forthcoming years. The CoAP covert channels presented here, are suitable for sending short messages, as our performance evaluation showed. Additionally, the performed experimental evaluation of power consumption analysis on one of the covert channels, shows only a slight increase in the power consumption of the used device, when sending two bits. The consequence of all these results, is the importance of identifying as much as it can, the possible ways of hiding data in CoAP and trying to mitigate them. One can deploy active and passive wardens for this purpose, but this is left for later investigation.

REFERENCES

[1] A. Mileva, A. Velinov, and D.Stojanov, "New Covert Channels in Internet of Things," Proc. 12th International Conference on Emerging Security Information, Systems and Technologies - *SECURWARE 2018*, Venice, Italy, 2018, pp. 30-36.

[2] Constrained RESTful Environments (CoRE) Parameters, CoAP Codes [Online]. Available at: https://www.iana.org/assignments/core-parameters/core-parameters.xhtml [retrieved: July, 2018]

[3] K. Denney, A. S. Uluagac, K. Akkaya, and S. Bhansali, "A novel storage covert channel on wearable devices using status bar notifications," Proc. 13th IEEE Annual Consumer Communications & Networking Conference, *CCNC 2016*, Las Vegas, NV, USA, 2016, pp. 845-848, doi: 10.1109/CCNC.2016.7444898.

[4] N. Feamster, M. Balazinska, G. Harfst, H. Balakrishnan, and D. Karger, "Infranet: Circumventing Web Censorship and Surveillance," Proc. 11th USENIX Security Symposium, San Francisco, CA, 2002, pp. 247-262.

[5] S. Fincher et al., "Perspectives on HCI patterns: concepts and tools," Proc. Extended Abstracts on Human Factors in Computing Systems (CHI EA '03). ACM, New York, NY, USA, 2003, pp. 1044–1045, doi: 10.1145/765891.766140.

[6] D. V. Forte, "SecSyslog: An Approach to Secure Logging Based on Covert Channels," Proc. First International Workshop of Systematic Approaches to Digital Forensic Engineering (SADFE 2005), Taipei, Taiwan, 2005, pp. 248-263, doi: 10.1109/SADFE.2005.21.

[7] M. Guri, G. Kedma, A. Kachlon, and Y. Elovici, "AirHopper: Bridging the Air-Gap between Isolated Networks and Mobile Phones using Radio Frequencies," MALWARE 2014, 2014.

[8] A. Houmansadr, N. Kiyavash, and N. Borisov., "RAINBOW: A Robust And Invisible Non-Blind Watermark forNetwork Flows," Proc. 16th Network and Distributed System Security Symposium (NDSS 2009), San Diego, USA, The Internet Society, 2009.

[9] M. N. Islam, V. C. Patil, and S. Kundu, "Determining proximal geolocation of IoT edge devices via covert channel," Proc. 18th International Symposium on Quality Electronic Design, ISQED 2017, Santa Clara, CA, USA, 2017, pp. 196-202, doi: 10.1109/ISQED.2017.7918316.

[10] B. W. Lampson, "Note on the Confinement Problem," Commun. ACM vol. 16, 10, Oct. 1973, pp. 613-615, doi: 10.1145/362375.362389.

[11] D. Martins and H. Guyennet, "Attacks with Steganography in PHY and MAC Layers of 802.15.4 Protocol," Proc. Fifth International Conference on Systems and Networks Communications (ICSCN), Nice, France, 2010, pp. 31-36, doi: 10.1109/ICSNC.2010.11.

[12] W. Mazurczyk and Z. Kotulski, "New Security and Control Protocol for VoIP Based on Steganography and Digital Watermarking," Annales UMCS Informatica AI 5, 2006, pp. 417-426, doi: 10.17951/ai.2006.5.1.417-426.

[13] W. Mazurczyk and K. Szczypiorski, "Steganography of VoIP Streams," in On the Move to Meaningful Internet Systems (OTM 2008) Robert Meersman, Zahir Tari (Eds.). LNCS, vol. 5332, 2008, pp. 1001-1018, doi: 10.1007/978-3-540-88873-4_6.

[14] W. Mazurczyk, S. Wendzel, Z. Zander, A. Houmansadr, and K. Szczypiorski, "Information Hiding in Communication Networks" Wiley / IEEE Comp. Soc. Press, (2016).

[15] A. Mileva and B. Panajotov, "Covert channels in TCP/IP protocol stack - extended version-," Central European Journal of Computer Science vol. 4, 2, 2014, pp. 45-66, doi: 10.2478/s13537-014-0205-6.

[16] A. K. Nain and P. Rajalakshmi, "A Reliable Covert Channel over IEEE 802.15.4 using Steganography," Proc. IEEE 3rd World Forum on Internet of Things (WF-IoT), Reston, VA, USA, 2016, pp. 711-716, doi: 10.1109/WF-IoT.2016.7845486.

[17] A. Nordrum, "Popular Internet of Things Forecast of 50 Billion Devices by 2020 Is Outdated," IEEE Spectrum. 18 August, 2016.

[18] R. Patuck and J. Hernandez-Castro, "Steganography using the Extensible Messaging and Presence Protocol (XMPP)," Computing Research Repository arXiv:1310.0524, 2013.

[19] Z. Shelby, K. Hartke, and C. Bormann, "The Constrained Application Protocol (CoAP)," RFC 7252, 2014.

[20] N. Tuptuk and S. Hailes, "Covert channel attacks in pervasive

computing," IEEE PerCom, pp. 236–242, 2015.

[21] X. Wang and D. S. Reeves, "Robust correlation of encrypted attack traffic through stepping stones by manipulation of inter packet delays," Proc. 10th ACM Conference on Computer and Communications Security (CCS'03), 2003, pp. 20-29, doi: 10.1145/948109.948115.

[22] X. Wang, S. Chen, and S. Jajodia, "Tracking anonymous peer-to-peer VoIP calls on the Internet," Proc. 12th ACM Conference on Computer and Communications Security (CCS'05), Alexandria, VA, USA, 2005, pp. 81-91, doi: 10.1145/1102120.1102133.

[23] S. Wendzel, " Covertand Side Channels in Buildings and the Prototype of a Building-aware Active Warden" IEEE ICC, pp. 6753-6758, 2012.

[24] S. Wendzel, S. Zander, B. Fechner, and C. Herdin, "Pattern-Based Survey and Categorization of Network Covert Channel Techniques," ACM Computing Surveys vol. 47, 3, Article 50, 2015, doi: 10.1145/2684195.

[25] S. Wendzel, W. Mazurczyk, and G. Haas, "Don't You Touch My Nuts: Information Hiding in Cyber-physical Systems," IEEE SPW 2017, pp. 29-34, 2017.

[26] S. Zander, G. Armitage, and P. Branch, "A survey of covert channels and countermeasures in computer network protocols," IEEE Communications Surveys and Tutorials vol. 9, 3, 2007, pp. 44-57, 10.1109/COMST.2007.4317620.

[27] Internet of Things technology [Online]. Available at: http://thingschat.blogspot.com/2015/04 /contiki-os-using-powertrace-and.html [retrieved: January, 2019]

[28] Z1 Datasheet [Online]. Available at: http://zolertia.sourceforge.net/wiki/images/e/e8/ Z1_RevC_Datasheet.pdf [retrieved: January, 2019]