An Implementation of Access-Control Protocol for IoT Home Scenario

Xiaoyang Wu, Ron Steinfeld, Joseph Liu, Carsten Rudolph

***Abstract*—The internet of things comes into our daily life. It connected lots of resource-constrained devices, denoted as smart device, in an Internet-like structure. Considering the computing burden, the CoAP protocol is developed for serving the resource-constrained device and maps to HTTP for integration with existing web. In this paper, an access-control protocol will be introduced. The protocol is designed for IoT(Internet of Things) home scenario. Like the most IoT we can see, the IoT home scenario contains lots smart devices which collect some private information from us. To protect those data, an access-control protocol is needed. The protocol is deployed into Contiki OS and evaluated using the powertrace and some other tools. The results shows the protocol we designed takes a little more memory usage than an OAuth based authorisation protocol but smaller power consumption and more suitable for small scale IoT environment.**

***Index Terms*—IoT(Internet of Things), protocol, Contiki, power consumption.**

I. Introduction

INTERNET of things (IoT) [1] has undoubtedly made great changes to the lives of people, ranging from intelligent furniture to smart city. IoT connected a large amount of devices and make them accessible remotely through Internet. While the IoT offer use great convenience, it also expose lots of private data on the internet.

In home scenario, IoT devices can help users access and control their appliances remotely. The connected appliances may record informations about the user. While lots of your valuable personal privacy information stored in IoT devices like monitor or camera, IoT devices may become target of malicious users. As all the IoT devices in home connected to the Wi-Fi network, an access-control protocol is necessary for protecting users privacy.

In the scenario that resource owner grants someone else to access resource, OAuth is very popular as it can authenticate the third-party without giving password. With the communication with an authentication server, the third-party keep a secret token to access users data. The scenario of OAuth used is similar to the IoT home scenario, but the power consumption and memory usage of OAuth is a great burden for IoT devices. So, a new access-control protocol is designed and implemented in our research for IoT home scenario.

As my partner have done the design of the protocol, my research is deploying the protocol he designed for home scenario to real machine. In the home scenario, most devices use low-power chips which only have low-power CPU, limited RAM and ROM. As the result of that, deploying the protocol to IoT devices, which are low-power chips, is most challenging part in the whole implementation. This research is to deploying the protocol (device part) to low-power chips.

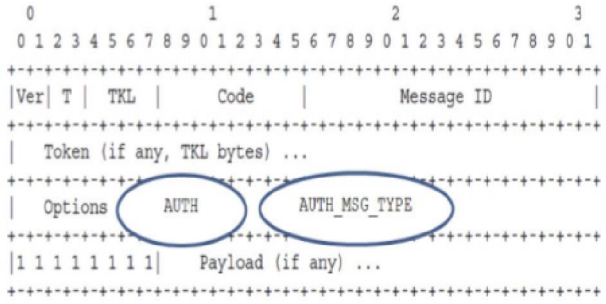


Fig. 1. Options added into CoAP header to implement security scheme[8]

II. Related works

A. Constrained Application Protocol

Constrained Application Protocol is a communication protocol based on REST model, which designed for low-power devices. It is a specialized Web transfer protocol for use with constrained nodes and constrained network. Because both CoAP and HTTP follows the REST model, they are very similar. [2] The server offer resources under a URL and client can access using the methods same as Http, e.g. GET, PUT, POST, and DELETE. As it designed based on REST model, it can be used as a common communication protocol which can transfer data between different devices in the IoT system.

CoAP is designed to be used on microcontroller with as low as 10 KB of RAM and 100 KB of code space. As my research is deploying the protocol to a low-power chip, a resource-saved protocol is really helpful.

A research about using CoAP to implement a lightweight security scheme for vehicle tracking system was done at 2013.They stated that the overhead security of CoAP is too high. [3] In their design, they use a payload embedded low cost symmetric-key based robust authentication and key management mechanism instead of handshaking and ciphersuite agreement of standard TLS and DTLS. For better performance in IoT system, they also modificate the header of CoAP to implement security special mechanism which better than the standard one.

To implement the CoAP with their security mechanism, they introduce a new option AUTH in CoAP header to enable the security function.

In their performance test, the security scheme they combined with CoAP only have a little effect on performance. The latency increase a little bit because of the header of CoAP become larger. It has a low overhead due to the authentication based on payload embedded symmetric key.

978-1-5090-5507-4/17/$31.00 ©2017 IEEE
ICIS 2017, May 24-26, 2017, Wuhan, China

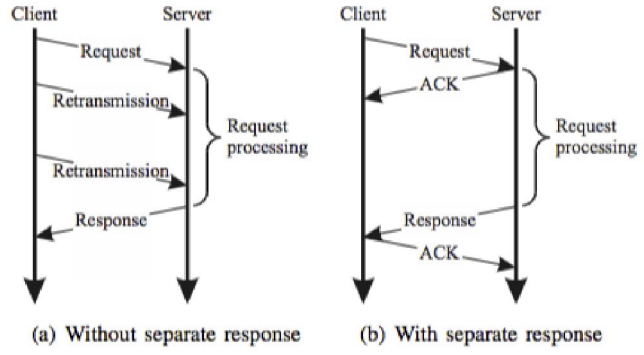


Fig. 2. Separate Response and ACK in CoAP[9]

Implementing the security scheme based on payload embedded content is an innovative ideal of modification of the CoAP. It can save resource which is really precious for devices in IoT system. In other way, it doesnt affect the performance of communication. Their research is for vehicle tracking system so that latency affect a lot. However, in home scenario, the latency is low because all devices connected to the same network without any long-distance transfer. So the disadvantage (increasing latency) of their design has no effect on our scenario. In performance aspect, this design fit our scenario exactly.

B. Low-power CoAP for Contiki

There is a research about implementing a low-power CoAP for contiki operating system in 2011. [4]. Their aim is to achieve high power efficiency for CoAP communication on Contiki platform. On typical IoT platform, one of components which are most power-consuming is the radio transceiver. The power consumption of listening packets is as expensive as receiving packets. [4] To reduce the waste of power, several radio duty cycling(RDC) algorithms have been designed. RDC algorithms allows node to turns off radio chip for most of time while still being able to send and receive message.

They also took advantage of the Contiki REST layer abstraction to modify the CoAP : they separate the acknowledge frame from response data. With this modification, the client doesnt need to resend the data during the time the server processing the request it receive. It enabling long processing times and avoiding unnecessary retransmission.

Their low-power CoAP implementation for Contiki achieve a high energy efficiency at the cost of a higher latency. The radio chip on server doesnt keep listening all the time, hence not all packets sent by client can be received and increasement of latency.

In out research, in home scenario, most IoT devices uses battery as power. High energy efficiency means the devices can keep working longer. This research shows a clear way to reduce the energy cost of devices. The energy efficiency may become a concern in further research but not in current state.

C. IOT-OAS

An OAuth-based authorisation service architecture is published in 2015 which called IoT-OAS. [5] The goal of the architecture is revlieving smart object form burden of handling large amount of authorization-related information. In the architecture, an external authentication service is set to handle authorization information. And the smart object can keep the application logic as simple as possible while outsourcing all the authentication functions.

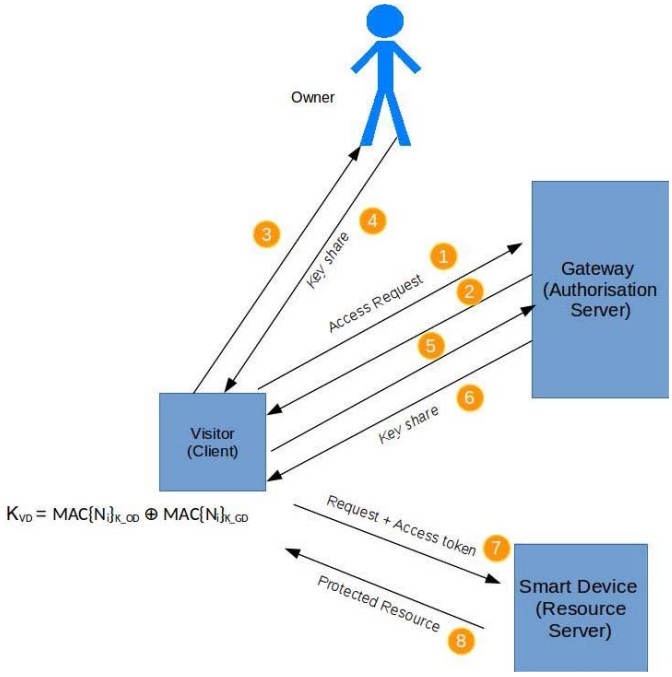


Fig. 3. Authentication Process

Considering the computing burden of using HTTP, they using different communication protocol among different roles in the architecture. The somart object only deal with message on COAP protocol. An hybrid gateway-based communication is used in their implementation. The application protocol extternal client used is different from that used by smart object. The gateway manage the communication in authentication process. It translate the incomming message from HTTP and CoAP to the other.

1) The particle swarm optimization algorithm:

III. Implementation

A. Protocol Summary

The protocol is designed by my partner. There are four roles in his design: Owner, Visitor, Gateway and Device. The most challenging of implementation is device part as the device program is working on power constrained chip. The protocol summary below is focus on the device program.

Authentication Process:The authentication process shows as Fig.3.

1 Visitor sends request about accessing device to gateway.
2 Gateway check the authority of the visitor in its database. If the visitor is eligible to access the device, send back the counter number and authorization information.
3 The visitor sends the counter number and authorization information to the owner.

4 The owner will check its local database. If the visitor is eligible to access the device, the owner will send back the Nonce, the $MAC_{K_{OD}}$, and the KeyshareO
Nonce = $HMAC_{K_{OD}}(counter)$
$MAC_{K_{OD}}$ = $HMAC_{KOD}(counter||AuthorizationInfo)$
KeyshareO = $HMAC_{K_{OD}}(Nonce)$

5 Then the visitor send the Nonce, to the gateway to obtain the KeyshareG.
KeyshareG = $HMAC_{K_{GD}}(Nonce)$

6 The gateway send back the KeyshareG. As KeyshareO and KeyshareG has been obtained, the K_{VD} can be calculate by visitor itself.
K_{VD} = KeyshareO‖KeyshareG

K_{VD} will be used to encrypt the request visitor send to the device. As AES128 encryption required the length of key must be times of 16, we take eight byte from each keyshare and combine them into a 16 byte key.

7 The visitor send the Authorization token and encrypted to the device
V to D : counter‖Authorization ‖
$MAC_{K_{OD}}(counter\|Authorization)\|$
$MAC_{K_{GD}}(counter\|Authorization)\|$
$\{Nv\|Request\}_{K_{VD}}$

The device processes this authorization token. Decrypted the request and send response if the authorization token is valid. The validation process on device will be described in detail in further section.

B. Communication

A gateway-based communication protocol is used in our implementation. There are two communication protocol we used under the same network. Considering the power limitation, we can not allow the IoT device running two server at the same time. To solve the problem, we set a HTTP-CoAP proxy on gateway. The data flows goes as Fig.4

The proxy is on the gateway. The packet owner/visitor send to device will go to the proxy first. Then the proxy translate it into a CoAP packet and send it to device. So does the packet send to owner/visitor by device.

C. Implementation Environment Summary

Gateway, owner and visitor supposes to be deployed to high-power environment: the gateway should be deployed to a router or a server which can running all the time. The owner and visitor program should be deployed to smartphone or desktop. In the home scenario, most devices use low-power chips which only have low frequencies CPU, limited RAM and ROM space. As the result of that, deploying the protocol to IoT devices is the most challenging part in the whole implementation.

D. Device Implementation

As I mentioned, the device is the only power constrained device in the system, so I focus on the implementation of device part program. All aspects of the implementation on device part will be described in detail in this section.

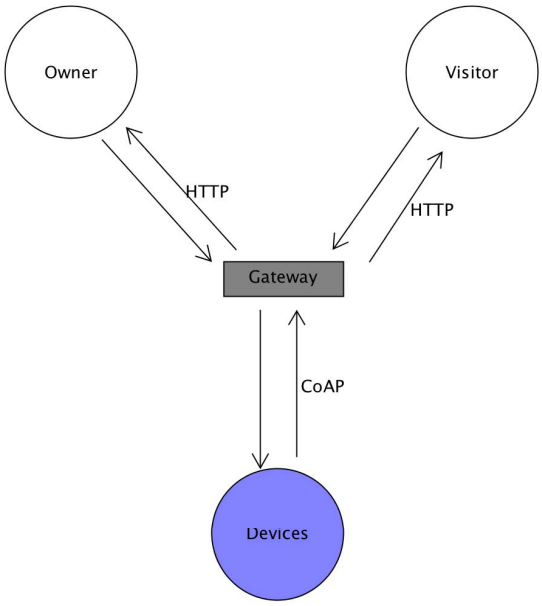


Fig. 4. Communication protocol in Implementation

E. Contiki OS

Contiki OS is an open source operating system designed for the Internet of Things. The protocol has been deployed to contiki-based constrained device. [6] The contiki operating system can runs on range of different platforms and is designed to run in small memory usage. Contiki support fully standard IPv6, IPv4, 6lowpan, RPL and CoAP. And contiki-based program can run on either the simulated environments (using cooja simulator) or real testbed.

The choice of Contiki-OS also allow us to do further investigation easily on the feasibility of the device part program.

F. Device Bootstrap

When the device boots, it will send its device id to gateway and owner. Each of them(owner and gateway) will generate a unique key basing on the device id they received. The key is generated by HMAC-SHA1 cryptograph. The device stores the key in RAM until reset. The owner and gateway also store the key in a database. After that, the device will have two synchronized key with the owner(K_{OD}) and gateway(K_{GD}). Also, the initial counter number will set by the gateway.

$$K_{OD} : HMAC_{Owner\_key}(Device\_ID)$$
$$K_{GD} : HMAC_{Gateway\_key}(Device\_ID)$$

Gateway and owner will use the key to authenticate visitor to access resources on the device. As keys are generated basing on device id, so keys are vary for different devices. We assume boot process running in a secure environment. After the device received two keys and counter number, it begin to listen to request from user.

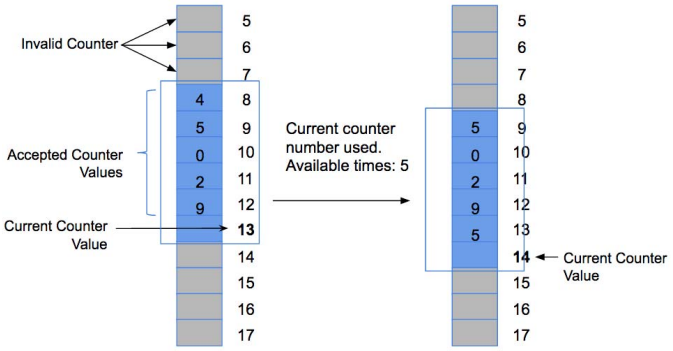


Fig. 5. Timer-based token expiration

G. Counter on Device

The access control is meaningless if an authorized token can be used forever. Most authorisation protocol has a timestamp which indicates when the authorisation will expire. In OAuth 2, a lifetime is used in authorisation token to indicate when the token expires. [7] Timer is the critical function module in the timestamp based key expiration. However, as target of the implementation is low power device, the CPU on chips may not contain timer on its own. The CPU without timer can not expire an authorisation token if token expiration mechanism is timestamp-based. Even the CPU has timer, more network transmissions are required for synchronize time with time server. For better portability of the device part program, the timestamp-based token expiration can not be used. A counter-based expiration mechanism is used on device program. The detail information of the mechanism can be seen in Fig.5.

Every device has its own counter. The length of counter is 4 byte, so the range of the counter number is from 0 to 2^{32}. When the device booting, the gateway would send a random initial counter number to the device. The device will set its current counter number to this initial counter number. In the Fig 5, the current counter number is 13. The default counter window size is 5, which means the counter number range from current counter number minus 5 to current counter number minus 1 is valid counter number. The device also keep an array of integer which named counterWindow. It record how many times left each valid counter number can be accessed. The index in array can map to specific counter number. Values in this array are times counter number their index map to can be used. The size of this array is exactly the size of available counter window. The logical structure of it can be seen in the Fig 5. For example, the counter number 8 times left is 4. And the number 12 times left is 9. Every times the counter number been used, the times left would minus one. When the times left becomes zero, like number 10 in Fig 5, it can not been used anymore. When current counter number been used for the first time, the counter number would increase. The access times left is setted by the last byte of authorisation information. As we can see in the Fig 5, the available counter window move down for one step. Current counter number becomes 14. And the access times for 13 is The old counter number 8 become invalid, even if it still has used times left. The number of

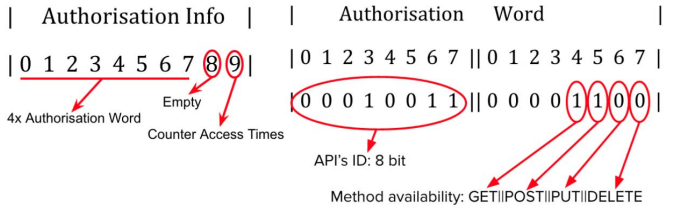


Fig. 6. Authorisation information structure

valid tokens at the same time is constrained by the size of counter window. For example, the default counter window is 5. So there is 5 tokens are valid at most. In home scenario, 5 valid token at the same time should be enough. It can also be customize basing on different need. In this mechanism, one more user at the same time only take one byte more in devices RAM. So the protocol can support large amount of visitor at the same time. The counter-based token expiration mechanism has one drawback. The token is exactly the same for the same counter number in same device. When the counter number goes to maximum, it will become zero in next time. So the counter number will finally go back to a number it used before if the counter didnt be reset. For prevent this kind of attack, we set the length of counter to 4 bytes. The counter number ranges from 0 to 2^{32}. And when device reboot, it will get a new random counter number from gateway. With these two configuration, the counter number became almost impossible to use some number it used.

H. Authorisation Information

The authorization information(10 bytes) contain four authorisation words and a byte to indicate how many times can the counter number be used.

The first eight bytes of the authorisation is made up of four authorisation words. The ninth byte in the authorisation information is empty. The last byte of the authorisation indicate how many times the counter number can be used. When the counter number in the packet is used for the first time, the counter number available times will be set on the devices. After that, the counter number available times can not be increased until reset.

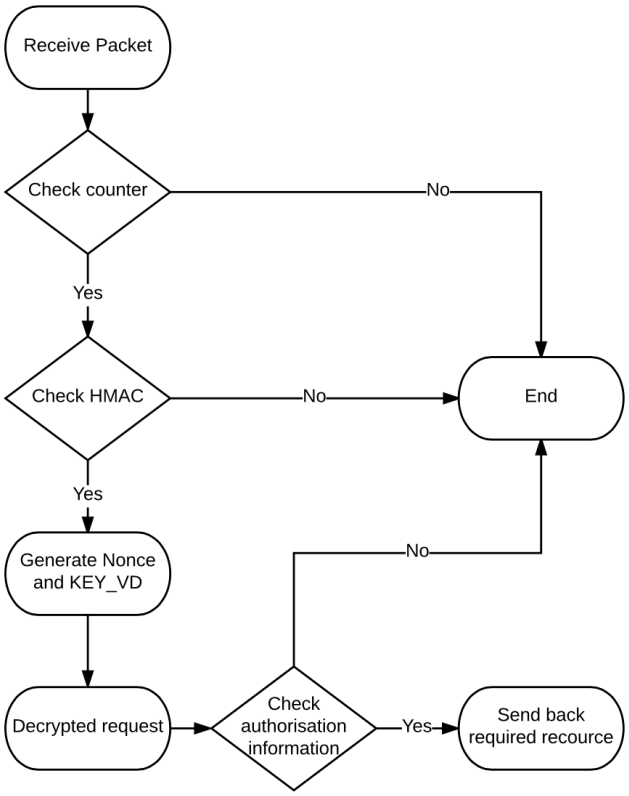
The length of authorisation word is 2 bytes. The first byte is the ID of API. Each API has its own ID, device declared ID of each API to the gateway when booting. The second byte denotes the approved methods for this API. As shown in Fig 8, the last four bits indicate the availability of methods GET, POST PUT and DELETE. In the Fig 8, the APIs ID is 19, and the approved methods are GET and PUT.

I. Nonce

The Monce is a 20 bytes hash code generated using the counter number as content and K_{OD} as key.

$$Nonce : HMAC_{K_{OD}}(Counter)$$

The K_{OD} is a synchronized key between owner and device so that device can generate this by itself. The nonce is used



in generating K_{VD}, and the K_{VD} will be used to decrypt the request sent by the visitor.

J. Request Validation Process

When device receives a request, the counter number will be checked at first step. The device will check if the counter number is in the valid counter window. If so, the HMAC checking will be done in next step.The device generates $MAC_{K_{OD}}$ and $MAC_{K_{GD}}$ using the HMAC-SHA1 cryptography and keys stored in RAM. Keys stored in RAM are synchronized with keys in gateway and owner, so the $MAC_{K_{OD}}$ and $MAC_{K_{GD}}$ should be exactly same as the MAC string in the packet. The packet passing the MAC checking denotes the packet has been authorized by both owner and gateway.
After that, the nonce (described in the last section) and K_{VD} will be generated. As K_{VD} is used to decrypt the request part, the length of it can only be 128, 192 and 256 bits. For reducing RAM usage of the program, the shortest one, 128 bits, is used in this protocol. The K_{VD} take eight bytes from each hash string generated from the nonce.
When device decrypted the request (using the K_{VD}), the requested resource can be seen. The device would go back to the authorisation information to check if the requested resource is approved in authorisation words. If its approved, then generate the response and encrypted it with AES128. The K_{VD} would be used as the encryption key, same as the received packet.

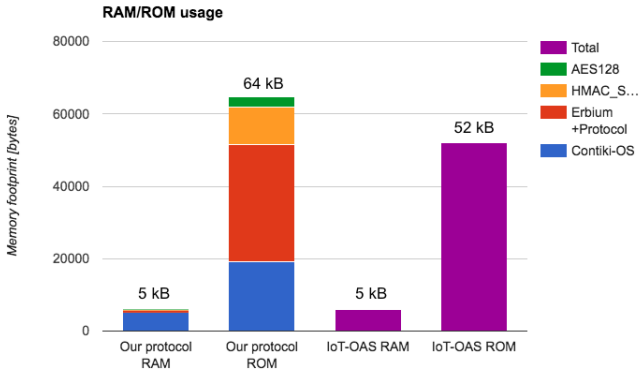


Fig. 7. Memory footprint, with compiler optimization enabled.

IV. Experimental Results

For demonstrating the feasibility and performance of the implementation of IoT access-control protocol designed by my partner, we have conducted experimental tests to evaluate the response time and energy consumptions on the device program. The security of the protocol is discussed by my partners thesis, so any attack model will not be tested in my research. In IoT home scenario, there are no power constraints on owner, visitor and gateway, so implementation of them are not critical from an energy consumption viewpoint. For this reason, we have limited our experiment to the device part program. The experimentation investigation include memory usage and energy consumption.

A. Experiment Setup

In the experiment, the device side program runs on a simulator called Cooja, which is the Contiki network simulator. It allows the network of contiki motes to be simulated. Mote is a hardware la emulation of low-power environment. Lots motes provided in Cooja and each of them has different RAM and ROM space configuration. Considering memory footprint of our program, the mote EXP5438 is used in our experiment. Cooja also offer lots tools which allow precise inspection of the system behaviour. Both network traffic and console output of the mote can be inspected in Cooja. Copper(Cu) is an add-on application of firefox browser. It allow users to manipulate a CoAP packet, send the packet and display the response. The Copper is used as a CoAP client to mock the visitor and gateway in our implementation. By sending CoAP packet to the device, the response time and energy consumption of processing a request can be inspected.

B. Memory Footprint

To evaluate RAM and ROM footprint of the device side program, the tool namedsize is used for get memory usage of the file after compile. The gnu size utility can list each section size for object or archive file in its argument list. By using this tool, the memory footprints of each module are obtained.

As shown in Fig.7, memory footprint of each component in the implementation has been measured. The complete device sied program take 64 kB in ROM and 5 kB in RAM totally. Among them, the Erbium combined with the protocol take most ROM space, which is 32 kB. The ROM footprint of Erbium can still be compressed some functions of it are turned on and useless in our IoT home scenario. In both RAM and ROM footprints, the Contiki OS takes up lots of memory space. The researcher of IoT-OAS declared that they turned off lots of function of Contiki OS to express memory footprints. [5] I havent tried it but this method should work on our project. The compiler optimization has been done when compiling. The command of the compiler optimization has been mentioned in related works.

Compared with the IoT-OAS implementation, their implementation only take 52 kB ROM space. As they didnt give the exact number of memory each module took, i can only compare the total memory footprint with them. They declared they have no resource on the Erbium CoAP server while our implementation has one test resource on the server. And we have two cryptographs(AES128 and HMAC_SHA1) on our server while they only has one on their server [5]. The memory compress approach they used will be try in our implementation in further research.

C. Response Time

As the console output can be inspected precisely through Cooja, the times each function takes can be calculated. By insert printf() command in the program, the timestamp this line be printed to console can be measured.

As we using printf() to inspect when each function completed, the times taken by printf() will affect the experiment result. So the time printf() taken needs to be measured at first. The time function printf() takes is measured by printing ten lines continuously. After calculating, the average times a printf() command takes is less than 1ms. As times it taken is too small, so the times printf() taken are ignored in experiments.

After experiment. The times of each component in the implementation has been measured as TableI.

TABLE I
ESTIMATED TIME FOR EACH MODULE.

Function part	Time
AES128 Decryption	5ms
HMAC_SHA1	10ms
Server+Protocol	(Response Time)94ms

D. Power Consumption Evaluation

The energy consumption of the CoAP has been evaluated by Powertrace [8], which is a tool for network-level power profiling in low-power wireless networks. The Powertrace can estimate energy consumption for each component on the board, such as the radio module and the microcontroller. The Powertrace record how long each component spend in specific model to estimate how much energy it consumpt. For example, there are two model for CPU(microcontroller), one is the working model, which shows as CPU in powertrace record. The other one is suspended model, which shows as LPM(Low power model) in its record. So does radio chip. The two models of radio chip is RX and TX. RX denotes the radio chip is listening while TX means the radio chip is sending message.

To determine energy consumption for each hardware component, the electrical specification (voltage and current) of each component is necessary. The Contiki OS website declared the mote exp5438 use TI cc2420 as radio chip. [9]And the mote exp5438 refers to the TI experimental board using MSP430F5438 as microcontroller. From the datasheet of MSP430F5438 and cc2420, we found that the electrical specification for both of them. In particular, the MSP430F5438 microcontroller consumes 2.5mA in active mode for ROM program and 0.5 uA in standby mode(low power mode). The cc2420 radio chip consumes 17.4mA in TX mode and 18.8mA in RX mode. [10], [11].

After obtaining all information for calculation, the power consumption can be calculated from following conversion formula:

$$E = \sum_{j \in M} i_j \cdot v \cdot \triangle t_j$$

Where M is the set of all operation modes of all hardware components on device(including cpu active, cpu LPM, RX, TX); i is the system current in each mode; v is the nominal voltage of the device; the t is the time the device stand on j mode.

The Fig.8 is the aggregate energy consumption evaluation for different protocol implementation on Contiki system. It shows how much energy will be consumed for processing one request in three different protocol. The aggregate energy consumption is broken down into energy consumption of CPU, radio transmission and low power mode. As the radio chip keep listening to radio signal, the RX mode consumes a great amount of energy every second. So the energy consumption of RX didnt be included in the Fig.8. In our evaluation, the RX mode (radio listening) consumed 28.5mJ per second.

The energy consumption of our protocol is 0.61mJ totally for processing one request. Among three modes shows in the Fig.8, the CPU active mode consumed most energy in the process because there are six cryptograph computing (5 HMAC_SHA1 hash computing and 1 AES128 decryption)are done in the process. As we only send one packet which is the response, so the TX mode doesnt consume much energy. As for LPM , our CPU is really busy when processing the request, so it is not much time left for low power mode. As a result of that, the power consumption of LPM can almost be ignored in the request processing.

The energy consumption data of other protocols are provided in the thesis about IoT-OAS [5]. So we can do a comparison of energy consumption between our protocol and theirs. The OAuth protocol consumes less energy than ours in processing one request. The power consumption of CPU active mode is much less than ours. This is reasonable as the OAuth token validation is much simpler than ours. And the

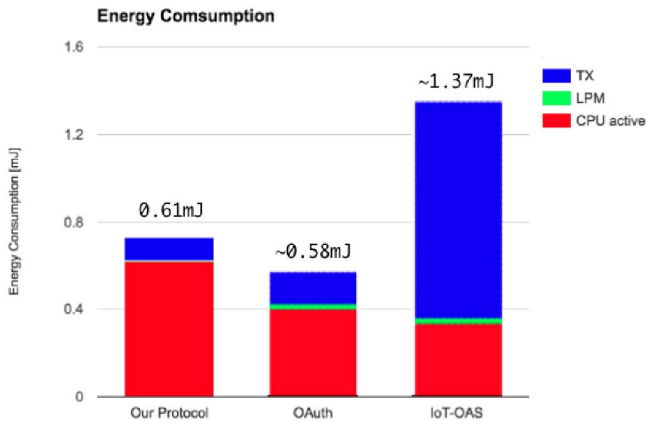


Fig. 8. Aggregate energy consumption

OAuth only sending one packet which is same as ours, so the power consumption of TX mode is similar as ours.

Comparing with the IoT-OAS protocol, ours CPU active mode consumes much more energy than theirs. Thats because they do the validation through another validation server. This mechanism make their device can only do a little part computing. But the price of less CPU computing is spending great amount of energy in communicating with the validation server, so their power consumption on TX mode is pretty high.

V. Discussion

In this paper, we have present an implementation for access-control protocol for IoT home scenario. According to the architecture of this implementation and the experimental result, the following aspects need to be discussed. The security issue will not be discussed in this paper because the design of the implementation will be discussed in my partner paper.

The advantage of this implementation is can support a large amount validate user at the same time. As the counter-based token expiration mechanism, the amount of validate user at the same time can be modified just by changing the size of the window. In OAuth, one more user may cost much more memory space for store the user information. Compared with OAuth, our protocol is more lightweight. In our protocol, the cost of adding one more user is only 1 byte in RAM footprint.

The memory footprint is the drawback of this implementation. As we can see in the memory footprint at experimental results, our implementation takes larger memory in ROM than other protocol. As some low-power chip can not offer such large ROM space, we can only deploy the implementation to some chip having large ROM. However, most chips with large ROM consuming more energy when CPU is activated. So the ROM size of the implementation affect the energy consumptions in some ways.

VI. Conclusion

In this paper, we proposed implementation of the protocol designed by my partner. The implementation details and some challenges in implementation has been described. The implementation approach has been applied to significant IoT home scenario device characterized by constrained memory space and limited computational power. The response time of the server is also considered and the experimental result shows the response time of the device is feasible.

In further research, we will try to compress memory footprint of Contiki OS and the Erbium CoAP server. With lower memory footprint, we can deploy our protocol to chips whose power is lower and make this implementation more feasible for IoT home scenario.

References

[1] F. Liu, "A survey of the internet of things," in *International Conference on E-Business Intelligence*, 2010.
[2] Z. Shelby, K. Hartke, and C. Bormann, "The constrained application protocol (coap)," 2014.
[3] A. Ukil, S. Bandyopadhyay, A. Bhattacharyya, and A. Pal, "Lightweight security scheme for vehicle tracking system using coap," in *International Workshop on Adaptive Security*, 2013, p. 3.
[4] M. Kovatsch, S. Duquennoy, and A. Dunkels, "A low-power coap for contiki," in *IEEE Eighth International Conference on Mobile Ad-Hoc and Sensor Systems*, 2011, pp. 855–860.
[5] S. Cirani, M. Picone, P. Gonizzi, and L. Veltri, "Iot-oas: An oauth-based authorization service architecture for secure services in iot scenarios," *IEEE Sensors Journal*, vol. 15, no. 2, pp. 1224–1234, 2015.
[6] "The contiki operating system," 2016, http://www.contiki- os.org.
[7] D. Hardt, "The oauth 2.0 authorization framework," 2012.
[8] A. Dunkels, J. Eriksson, N. Finne, and N. Tsiftes, "Powertrace : Network-level power profiling for low-power wireless networks low-power wireless," *Swedish Institute of Computer Science*, 2011.
[9] "Contiki hardware," http://www.contiki-os.org/hardware.html.
[10] T. Instruments, "Cc2420 datasheet," *Reference SWRS041B*, 2007.
[11] T. Instruments, "Msp430f5438 datasheet," *Reference SLAS655B*, 2010.