CLI-DOS: Collaborative Counteraction against Denial of Service in the Internet of Things

Syafiq Al Atiiq*, Christian Gehrmann*
*Department of Electrical and Information Technology, Lund University
Lund, Sweden
{syafiq_al.atiiq, christian.gehrmann}@eit.lth.se

***Abstract*—Internet of Things (IoT) is the next generation of network scenario where billions of devices are connected to the internet and capable to communicate with each other. However, they are especially vulnerable to Denial of Service (DoS) attack, which typically launched by overloading the server with bogus messages. Hence, unnecessary computation is performed by the IoT units, which can seriously worsen their performance or even make them unable to serve legitimate requests. Existing mechanisms for the detection and prevention of DoS attacks only either make it more difficult to perform massive DoS attacks or completely block the legitimate requests to be served. In this paper, we present CLI-DOS, a collaborative mechanism between Gateway and IoT units to counteract Denial of Service attacks. CLI-DOS combines an efficient attack-detection principle using short MAC value with a more effective defense strategy. In an under attack situation, we leverage the IoT gateway for preventing the attack messages to make contact with the IoT units while opening up possibilities to set-up connection with legitimate external entities. We evaluate the performance of our proposal from several standpoints. In conclusion, CLI-DOS offers a lower resource utilization in an under attack IoT units while preserving service availability to the best possible extent.**

***Index Terms*—Internet of Things, Denial of Service, Security.**

I. Introduction

Internet of Things (IoT) is commonly referred to an increasing trend in information technology en route to *networked society*, where all devices are interconnected and able to harness the connection to the internet [1]. In some cases, these devices are even used for prominent and important tasks such as health monitoring, industrial automation, or energy management. However, many IoT devices are built upon a constrained hardware platform, which has limited resources of CPU, memory, or even power source compared to e.g. personal computers. That being said, keeping the availability of the devices as long as possible becomes a very important task.

IoT units with a direct connection to the internet are prone to be exposed to DoS attack. The common strategy used by the adversaries consist in flooding the server devices with invalid messages such that the server becomes exhausted and unable to serve legitimate requests. As most IoT units are battery-powered, the impact of DoS attack to those devices is considerably severe which can greatly reduce the lifetime of the IoT units or even make them out of battery at once.

A number of countermeasure strategies have been proposed to counteract DoS attacks. As explained by Wang [2], strategies can be classified into two types, namely *router-based* and *host-based*. First, *router-based* approach addresses the problem by deploying a built-in defense mechanism running on the intermediate nodes between client and server. On Cisco routers [3], this include for instance: reverse routing path verification and IP address filtering. As for the *host-based* solution, IoT units are designed to use cryptographic protocols e.g. DTLS [4] and IKE [5]. As these protocols use a stateless *cookie* exchange, launching DoS attack with spoofed IP addresses towards IoT units would be difficult. On the other hand, the cryptographic operations employed in the IoT units can be quite expensive in terms of power consumption at the set-up phase. Furthermore, as mentioned in the standard [4], a *cookie* does not provide any defense mechanism against DoS attack launched from valid IP addresses.

Another approach [6] is by forcing the adversaries to solve a computational demanding puzzle as a requirement to proceed to the handshake phase. However, with an increasing power of modern GPU, the puzzles from the victim can be easily solved [7]. Furthermore, a more advanced strategy to determine the validity of incoming request is by including a short symmetric key based MAC value in the packet header when both parties, client and server, are trying to establish a connection [8] [9] [10]. While this is very efficient to detect the occurrence of DoS attack in the IoT units, such strategy will at the same time completely prevents the legitimate clients to reach the IoT units. An adaptive solution, called SARDOS [11] has been proposed to counteract DoS attack while preserving service availability from legitimate clients. Even though SARDOS mechanism is already good enough to prevent the battery drain by adaptively changing the server state, but the computational burden to check the validity of the incoming requests is still inside the IoT units. Aside from shutting down the interface, there is no mechanism in SARDOS to drop the attack messages even before reaching the IoT units.

To fill the gap, in this paper we introduce CLI-DOS, a collaborative mechanism between the IoT units and the gateway to counteract DoS attacks. In particular, we leverage an efficient DoS detection mechanism from the previous work

Work supported by framework grant RIT17-0032 from the Swedish Foundation for Strategic Research as well as the EU H2020 project CloudiFacturing under grant 768892

[8] [9] [10] with a novel collaboration protocol between the IoT units and the gateway at the border of IoT wireless domain. Our rationale is, it would be better to offload some computational expensive process of filtering attack messages from the IoT units to a much more powerful gateway that sits close to the IoT wireless domain. The gateway will have the capability to filter the message ID of the CoAP request based on the acceptable range provided by the victim. This way, we are able to protect the wireless IoT units from battery drain attack, even when the adversaries changing their IP address many times, *while at the same time* allowing legitimate request to be served. CLI-DOS will to some extent lessen the capabilities of IoT units to communicate with external entities, however, it will not completely block the communication from serving the critical messages even when under heavy attacks.

We have made a proof-of-concept implementation of CLI-DOS for the CoAP [12] protocol using ContikiOS [13] inside resource-constrained CC2538 platform. Yet, CLI-DOS is designed with the general mindset and can be implemented at any communication protocol stack. On the gateway side, we have made an implementation using $u32$ module of netfilter [14] inside the kernel in a way that it can parse the CoAP message ID as one of our requirement. Also, the design of our protocol in the gateway side is general so that it can be implemented on any linux flavor, either in *user-space* or *kernel-space*. As some computational burden of filtering attack messages has been offloaded to the gateway unit, our results show that CLI-DOS has a lower resource utilization in terms of energy consumption at the IoT units compared to the vanilla CoAP implementation and pure IP blocking mechanism at the gateway. In addition, CLI-DOS has efficiently dropped the bogus messages even before reaching the under-attack IoT units while still serving legitimate requests from external entities at the same time.

The rest of the paper is organized as follows. We discuss related works in Section II and background concepts in Section III. We provide the application scenario in Section IV. Section V presents collaborative solution against DoS, while in Section VI we provide a performance evaluation of CLI-DOS. Section VII draws our conclusions and anticipate future works.

II. Related Work

Denial of Service is a security menace, as defined at [8], in which the aim is to (partially or completely) disrupt the service of server(s). The victim is usually forced to use its resource in a high fashioned utilization. This include for instance CPU, memory, and network bandwidth as a means to make it less responsive or even unable to serve legitimate request. The use of DoS to attack can be classified into *Distributed Denial of Service* [15] if the attacker sources are using many machines. This usually done by creating a well coordinating master and slave nodes, in which the master act as a leader.

To counteract DoS, there have been several ideas proposed. As classified by Wang [2], there are two types of DoS counteraction, namely *router-based* and *client-based*. A router-based mechanism solve the problem by identifying address passed by the attacker [16], thanks to its built-in mechanism planted on every router on the path from the client to server. This particularly efficient for a network that is managed by single entity where we have the access to all the router. On the other hand, router-based solution normally need the involving router to implement *Probabilistic Packet Marking (PPM)* and coordinating mechanism between them. This would lead to the inconvenience of the person in charge as they need to maintain many coordinating nodes with additional complexity. Seidel proposes a protocol validation to complicate remote DoS attack [17]. While this is a good starting point to offload the filtering mechanism to the border router, we believe that a further step to determine the validity of request message should be taken into account.

Host-based approach counteract DoS by building a mechanism at the end nodes, which does not need a coordination function between entities. Several examples of *host-based* approach can rely on resource management schemes [18] [19]. Another proposed solution that can be classified into *host-based* leverages puzzle [20]. In order to legitimately access the server, a client has to solve puzzles which has several level of difficulty. While this can prevent a DoS to some extent, but with an increasing power of modern GPU, a puzzle from a resource constrained device can easily be solved [7].

Our solution in this paper falls between *router-based* and *host-based*. We leverage the simplicity of setting up *host-based* approach while still maintain contact with the router to get better quality and resource to filter bogus messages. To get a better segregation on valid and invalid messages on the server side, we leverage SMACK [9], a host-based countermeasure to DoS that make use of short Message Authentication Code (MAC). The server analyze the embedded short MAC on the request message sent to them and instantly get a classification of the validity of the message.

The additional value of our proposed solution compared to traditional DoS mitigation relies on the coordination function between victim server and the router on the same network. As the router is usually more powerful in terms of resource, we move some computational burden of filtering messages to them, not only using IP based blocking but also a mechanism based on message ID of CoAP message. This way, even the adversaries changing its IP address, we can preserve the battery on the server side while maintaining the service to the legitimate user to the best possible extent.

III. Background

A. CoAP

The Constrained Application Protocol [12] is a lightweight communication protocol, specifically designed for resource constrained nodes and networks. To some extent, it is similar to HTTP but for affordable and lightweight type of communication. CoAP's main purpose is to be used for Internet of Things environment, where machine to machine communication take place. Home automation, factory monitoring, and connected vehicles are some examples of where CoAP is suited.

Unlike HTTP, CoAP uses UDP as its transport protocol, meaning that it has less overhead in terms of the amount

of messages exchanged between parties. With regards to security, CoAP does not provide any mechanism, but rather offload it to other components. The most recommended one is DTLS, where it can provide secure communication, e.g. authentication, integrity, and confidentiality of the messages that's exchanged between nodes.

B. SMACK

SMACK [9], correspond to Short Message Authentication ChecK, is a security mechanism to identify a validity of incoming messages in the server. The way SMACK works is by probing into a short and lightweight Message Authentication Code embedded in the message. The output of SMACK processing is a verdict whether a message is a valid request (and proceed for further processing) or not (discarded).

SMACK is designed to work well with CoAP, in which it uses the Token field to embed short MAC in the message. By message structure, there is not much modification of the CoAP header, except the Token field itself. In advance of sending the message, client generates 16-bits random *Request ID R* as a means to match request and respective response messages. Following *R*, client computes 16-bits *short MAC SM*. Then, the client put *R* and *SM* in the Request ID and Validity check field respectively, as depicted in figure 1.

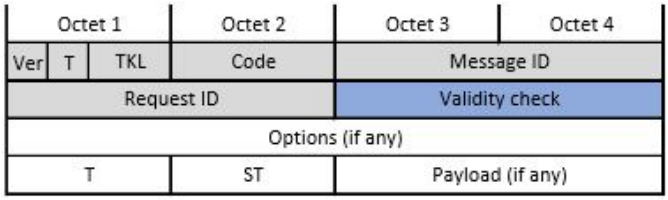

Fig. 1. CoAP Message with Short MAC

Upon receiving the message, the server computes the short MAC of the incoming message and compare the value with the provided short MAC of the request message.

IV. Application Scenario

Henceforward, we examine the scenario in Figure 2. It consists of a Server S and a Client C exchanging message over CoAP protocol. At the same time, an adversary A launch a DoS attack by sending large number of invalid CoAP messages towards S. The aim of the attack messages is to make S worthlessly commit resources in a way that is not serving actual legitimate request. A also has the capability to spoof its IP address as many times as he wants. This way, S could be endangered by becoming less responsive or even running out of energy due to battery drain.

A Gateway G sits between C and S. The gateway carry out the job as a message forwarder between C and S. We assume that the connection between G and S are in the wireless communication channel, meanwhile communication towards C is a wired connection. The adversary A always come from the wired connection domain. The case where A is coming from the wireless domain is out of scope of this paper.

At any times, S should be able to identify legitimacy of the request message. A legitimate request from C is always

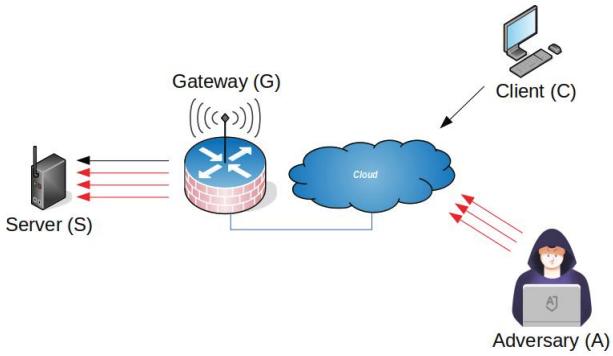

Fig. 2. Application Scenario

processed and replied with a CoAP response. On the other hand, an illegitimate request from A should be detected and S should further act (i.e. report to Gateway G) as explained in chapter V. In this paper, the capability to identify legitimacy of the message come from the implementation of SMACK [9].

We assume G is secure enough, such that it is unattainable to be compromised. It is possible to design G as a distributed gateway to get a better reliability as described in [21] [22]. The work to design such system is out of scope of this paper. Nevertheless, we assume that G is robust and reliable such that no matter how high the DoS magnitude, does not kill G.

V. Collaborative Counteraction Against DoS

In this section, we present CLI-DOS, the Collaborative Counteraction against DoS. The fundamental motivation of CLI-DOS lies in two following arguments: (i) It is possible to offload computational expensive filtering at the server to a much more powerful gateway, while at the same time (ii) Allowing legitimate request to be served in a best effort manner. CLI-DOS leverages the Message ID field in the CoAP header to determine the validity of subsequent CoAP requests. This will allow the gateway to detect IP spoofing events from the adversaries. CLI-DOS will to some extent reduce the communication capabilities of the server, however it will not completely stop the server to perform critical communication task even under heavy DoS attacks. The following subchapter explain the detailed procedures of CLI-DOS both from the server and gateway side.

A. Procedures from The Server Perspective

At any times, the server measures the number of invalid message(s) m for every time period t. In a normal operation, if there is no DoS attack occured, S resets m to 0 when t is over. However, if the number of invalid messages m exceeds a predefined threshold h, S contacts G and sends a request r to start DoS protection. Let ID_{qa} be the message ID accepted by SMACK as a valid request [9], and $\overline{ID_q}$ as a set of accepted message ID of size x, then Algorithm 1 summarize S procedures for every time period t.

When a newer valid request arrived, $\overline{ID_q}$ should be updated accordingly. For example, let x be 20, once ID_{q1} arrived, $\overline{ID_q}$ would contains $\{ID_{q2}, ID_{q3}, ID_{q4}, ..., ID_{q21}\}$.

Algorithm 1 Procedure at S

$\overline{ID_q} = \{ID_{q1}, ID_{q2}, ..., ID_{qx}\}$
$m = 0$
while time $< t$ **do**
  <Receive and check short MAC>
  if CoAP Request Valid **and** $ID_{qa} \in \overline{ID_q}$ **then**
    $\overline{ID_q}$.delete(ID_{qa})
    if $a < (x/2)$ **then**
      $\overline{ID_q}$.append($ID_{q(x+1)}$)
    else
      $\overline{ID_q}$.delete($ID_{q1}, ..., ID_{q(a-(x/2))}$)
      $\overline{ID_q}$.append($ID_{q(x+1)}, ..., ID_{q(x+1+(a-(x/2)))}$)
    end if
  else if CoAP Request Invalid **then**
    $m = m + 1$
    if r has not been sent **and** $m > h$ **then**
      <Send r to G, starts s_1 and turn off radio>
    else
      <Send m_f to G >
    end if
  end if
end while
$m = 0$

Algorithm 2 Procedure at G during attack period

<Receive r from S >
$\overline{ID_q} = \{ID_{q1}, ID_{q2}, ..., ID_{qx}\}$
<Starting timer s_2 >
<Apply filtering mechanism as requested by S >
$a = 0$ **and** $d = 0$
<Initiate messages buffer with the length x >
while s_2 is not expired **do**
  <Receive CoAP request packet from C >
  if $ID_{qa} \in \overline{ID_q}$ **then**
    $< a = a + 1$ **and** put the packet into buffer>
  else
    $< d = d + 1$ **and** discard the packet>
  end if
end while
if $a < p$ **then**
  <Forwards all queued packet to S >
else
  <More severe attack happens>
  <Sends warning w to the system responsible>
end if

Since CoAP runs on top of UDP, it is possible that request messages arrived out of order. For example, instead of ID_{q1}, it is ID_{q13} whose arrived first. Then $\overline{ID_q}$ would delete ID_{q13} and $\{ID_{q1}, ..., ID_{q(13-(20/2))}\}$, and append $1 + (13 - 20/2)$ additional message ID's at the end of the list. Hence, $\overline{ID_q}$ would become $\{ID_{q4}, ID_{q5}, ..., ID_{q12}, ID_{q14}, ..., ID_{q23}, ID_{q24}\}$. However, if the message ID is less than $x/2$, $\overline{ID_q}$ will just delete ID_{qa}, and append 1 new message ID at the end of the list.

Request r includes the following information:

- The latest state of set $\overline{ID_q}$.
- A list of recorded source IP addresses in which the Server S wants to block.
- Maximum number of allowed IP packets per second, p.
- Threshold value h, denotes the maximum number of invalid messages for a time period t.
- A sleep period, s.

Once S goes into the sleep mode, it initializes an internal clock s_1, which then followed by turning off its radio communication. On the other hand, if r has been sent to G, S only sends m_f, a flag message informing G of each individual invalid request arrived at S.

B. Procedures from The Gateway Perspective

When there is no request r from S, G operates normally by forwarding all request messages from C to S. However, when r received, G initialize s_2, a timer to indicate when S is going to wake up again. This step marks G to operates on a more strict rules on who can send CoAP request towards S.

During this period, all packets which have the destination address of S are checked with respect to their higher level protocol, such that: i) all packets with the messages ID not in the list of $\overline{ID_q}$, ii) all packets with forbidden IP addresses, and iii) all packets but CoAP, are dropped.

G measures the number of dropped and accepted packets per second targetting S, denoted as d and a respectively. When period s is over, S turned on its radio communication again, and notify G. When G get notified by S, it checks whether $s_2 > s$ as well as if $a < p$, where p is a threshold determines the maximum allowed load on S. If both conditions are met, G starts forwarding sequentially all queued packets targetting S until the packet buffer is empty. G then sets a to 0 and reinitialized s_2 to 0. Otherwise, when $s_2 > s$, G would assumes that a more severe attack happens and sends a warning message w to the system responsible. Algorithm 2 summarizes G behavior during the attack period.

Also, when filtering mechanism kicked in, G should have a MID list processing, regardless of the state of S. When a request with MID ID_{qa} received, G checks if it is in the $\overline{ID_q}$ or not. If yes, then G puts ID_{qa} into a temporary list $\overline{ID_t}$ where the size is adjustable. Meanwhile, G initialize a timer t_w, where it expects to receive an ominous message from S, saying that CoAP request with ID_{qa} is an attack. If it does not happen until t_w expired, G assumes that the request is valid, and delete ID_{qa} from $\overline{ID_t}$, followed by the adaptation of $\overline{ID_q}$ similar to the process explained in section V-A. Algorithm 3 summarizes the G message ID processing, which running parallel with Algorithm 2 as a non-blocking operations.

VI. Experimental Evaluation

In this section, we discuss the setup, scenario, and results of the evaluation of CLI-DOS. We compare CLI-DOS with Vanilla CoAP and pure IP based protection at G. We have developed a prototype of CLI-DOS on Contiki Operating

Algorithm 3 Procedure at G at any times

$\overline{ID_t} = \{\}$
<Receive CoAP with MID ID_{qa} from C >
if $ID_{qa} \in \overline{ID_q}$ **then**
  $\overline{ID_t}$.append(ID_{qa})
  <Starts t_w >
  if t_w expired **then**
    <CoAP message with ID_{qa} is valid>
    $\overline{ID_t}$.delete(ID_{qa}) **and** $\overline{ID_q}$.delete(ID_{qa})
    if $a < (x/2)$ **then**
      $\overline{ID_q}$.append($ID_{q(x+1)}$)
    else
      $\overline{ID_q}$.delete($ID_{q1}, ..., ID_{q(a-(x/2))}$)
      $\overline{ID_q}$.append($ID_{q(x+1)}, ..., ID_{q(x+1+(a-(x/2)))}$)
    end if
  else if m_f received before t_w expired **then**
    <CoAP message with ID_{qa} is an attack>
    $\overline{ID_t}$.delete(ID_{qa})
  end if
else
  <Drop CoAP Request>
end if

System [13]. The new developed firmware is tested on Zolertia Firefly platform [23], which has the following specifications : CC2538 radio chipset, 32 kB RAM, 512 kB of flash ROM.

Our experiment follows the scenario depicted in Fig 2, where S is connected to G using IPv6 over IEEE 802.15.4 [24] transmission running 6LoWPAN stack. S is the Zolertia Firefly [23] device running extended CoAP server equipped with CLI-DOS. G is a dedicated VM running contiki rpl-border-router and connected to a slip radio through serial connection. The combination of rpl-border-router and slip-radio makes it possible to bridge the wired domain with the 6LoWPAN protocol in the wireless domain. G, C, and A runs on a native linux and performs the computation on an identic but fully isolated VM with native IPv6 connectivity. All of the VM's equipped with 2 GB Memory and 1 core vCPU.

We perform three experiments, namely 1) E_PLAIN, where S running Vanilla CoAP implementation, and G acts as a forwarder, where all the packets are forwarded to S. 2) E_SMACK_IP, where S has SMACK [9] and able to shuts down its interface and report to G the source IP of A, when certain limit of attack is reached. For simplicity, we made the limit same as h value at E_CLIDOS_MID. 3) E_CLIDOS_MID, where S running CLI-DOS with message ID filtering mechanism. To this end, aside from source IP filtering, G has the capability to inspect CoAP message based on its MID, as we explained in the previous section. One experiment performed by running 100 Request-Response exchange messages between C and S. One experiment considered as done only if the last request message has been replied by the S, and received by C. It is possible to lost the message if S is fully occupied by A. In this case, the experiment considered to be done when the last retransmission message on the last request has been transmitted. C sends similar request message every 2 sec, with the size of 28 bytes.

We examine CLI-DOS performance with five different attack settings, that is: 0, 5, 10, 15, and 20 msg/s. We run each experiments 5 times for each attack settings, and then calculate the mean value. A sends 19-bytes CoAP request as the attack messages in all experiments. Also, during our evaluation, we set the threshold h to be 9 msg/s and sleep time s to be 60 sec. These values are determined heuristically thorugh repeated test and evaluations. The performances are measured in terms of i.) energy consumption at S, and ii.) RTT from C.

A. Results

Fig 3 shows average energy consumption at S for 5 different attack settings. When there is no attack, each of the experiments shows little difference on the energy usage. It means that SMACK as a message validator, does not have considerable amount of additional overhead.

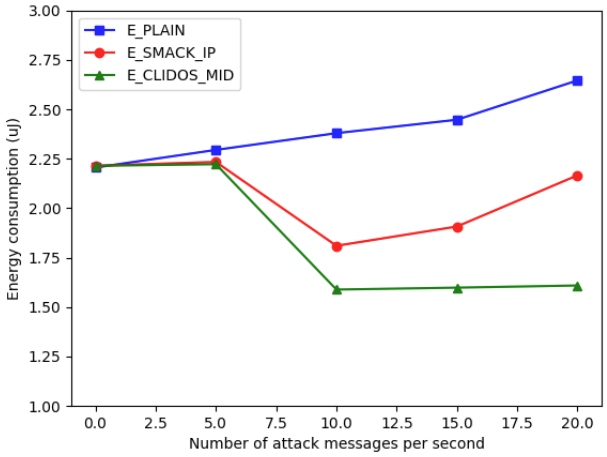

Fig. 3. Energy Consumption of each attack configuration

However, when the attack continue to rise to 5 msg/s, E_PLAIN uses 2.29 μJ, E_SMACK_IP uses 2.23 μJ, while E_CLIDOS_MID uses 2.22 μJ. Intuitively, E_SMACK_IP and E_CLIDOS_MID should show lower energy consumption than shown in the graph. However, this is happened because S considers 5 msg/s as a non-malignant attack. The reason is because we set the threshold h to be 9 attack messages per second (we would get a similar result with a different threshold selection). Hence, S would not send any request r towards G to start filtering protection. At 20 msg/s, E_PLAIN reaches the highest point 2.65 μJ, while E_SMACK_IP uses 2.17 μJ. On the other hand, E_CLIDOS_MID comfortably uses only 1.61 μJ energy during the experiment. It means that even though E_SMACK_IP performs IP filtering at G, it is not enough to protect S once A spoof its address. Meanwhile, as E_CLIDOS_MID performs deeper payload inspection (based on MID) for each incoming request, it has better performance because less bogus messages are arrived at S.

Fig 4 shows the average RTT for each different attack settings in all experiments. We can see that during 0 and 5 msg/s (benign attack), there is not much different

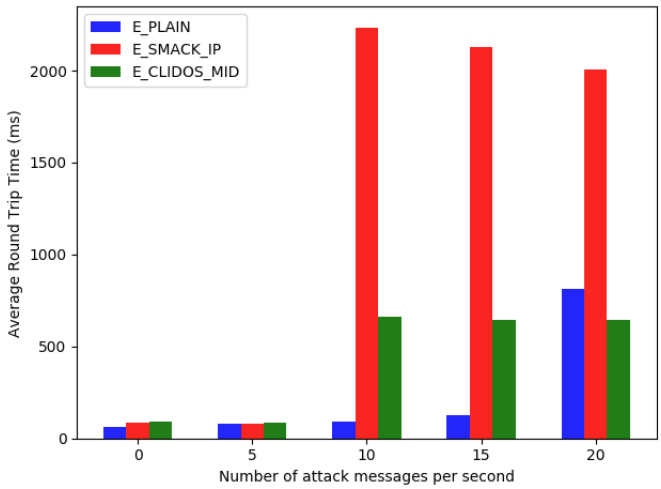

Fig. 4. Average Round Trip Time

between them. However, when the attack goes up to 20 msg/s, E_CLIDOS_MID (645.28 ms) performs better than E_SMACK_IP (2005.95 ms) and E_PLAIN (811.28 ms). E_SMACK_IP took longer time because it shuts down its interface much longer than E_CLIDOS_MID due to IP address spoofing at the advesary, which then followed by subsequent attack. On the other hand, E_CLIDOS_MID could serve the request better than others, due to MID blocking at G such that adversary A could not reach S even by spoofing their IP address. As Fig 4 is average value, E_SMACK_IP and E_CLIDOS_MID's are get affected by sleep period s of the victim. But, it is only a fraction of the messages need to wait longer to get reply due to s.

VII. Conclusion and Future Work

This paper has presented CLI-DOS, a Collaborative Counteraction against Denial of Service. Through this collaboration, it is possible to off-load the computational expensive packet filtering from the IoT units to a much more powerful gateway. This way, CLI-DOS has effectively prevent the IoT units from worhtlessly commit of resources during DoS attack, while at the same time serving legitimate requests to the best possible extent. The solution works without requiring any trust between the IoT unit and gateway except for sharing filtering parameters, i.e. SMACK shared secrets are *not* shared. We have experimentally evaluated CLI-DOS through our prototype on Contiki OS running on Zolertia Firefly platform. The results show that, when uder heavy attack, CLI-DOS successfully preserve the energy consumption, such that IoT units does not have to deal with the attack messages and can serve the legitimate client better. This come at the price of some longer RTT for some request packet, due to waiting time of the server when they shut down their interface. Future works will include evaluation of larger IoT networks and under different wireless channel conditions. Also, scenarios with more powerful adversaries is left to consider.

References

[1] G. Kortuem, F. Kawsar, V. Sundramoorthy, and D. Fitton, "Smart objects as building blocks for the internet of things," *IEEE Internet Computing*, vol. 14, no. 1, pp. 44–51, Jan 2010.

[2] H. Wang, C. Jin, and K. G. Shin, "Defense against spoofed ip traffic using hop-count filtering," *IEEE/ACM Transactions on Networking*, vol. 15, no. 1, pp. 40–53, Feb 2007.

[3] Cisco. (2008) Cisco guide to defending against distributed dos attacks.

[4] E. Rescorla and N. Modadugu, "Datagram transport layer security version 1.2," Internet Requests for Comments, RFC 6347, January 2012. [Online]. Available: http://www.rfc-editor.org/rfc/rfc6347.txt

[5] C. Kaufman, "Internet key exchange (ikev2) protocol," Internet Requests for Comments, RFC 4306, December 2005. [Online]. Available: http://www.rfc-editor.org/rfc/rfc4306.txt

[6] R. Hummen, H. Wirtz, J. H. Ziegeldorf, J. Hiller, and K. Wehrle, "Tailoring end-to-end ip security protocols to the internet of things," in *2013 21st IEEE International Conference on Network Protocols (ICNP)*, Oct 2013, pp. 1–10.

[7] Nithun Chand O and S. Mathivanan, "A survey on resource inflated denial of service attack defense mechanisms," in *2016 Online International Conference on Green Engineering and Technologies (IC-GET)*, Nov 2016, pp. 1–4.

[8] M. Tiloca, C. Gehrmann, and L. Seitz, "On improving resistance to denial of service and key provisioning scalability of the dtls handshake," *International Journal of Information Security*, vol. 16, no. 2, Apr 2017.

[9] C. Gehrmann, M. Tiloca, and R. Hoglund, "Smack: Short message authentication check against battery exhaustion in the internet of things," in *2015 12th Annual IEEE International Conference on Sensing, Communication, and Networking (SECON)*, June 2015, pp. 274–282.

[10] C. Gehrmann and G. Selander. (2013) Methods, nodes and computer programs for reduction of undesired energy consumption of a server node.

[11] M. Tiloca, R. Hoglund, and S. Al Atiiq, "Sardos: Self-adaptive reaction against denial of service in the internet of things," in *2018 Fifth International Conference on Internet of Things: Systems, Management and Security*, Oct 2018, pp. 54–61.

[12] Z. Shelby, K. Hartke, and C. Bormann, "The constrained application protocol (coap)," Internet Requests for Comments, RFC 7252, June 2014. [Online]. Available: http://www.rfc-editor.org/rfc/rfc7252.txt

[13] A. Dunkels, B. Gronvall, and T. Voigt, "Contiki - a lightweight and flexible operating system for tiny networked sensors," in *29th Annual IEEE International Conference on Local Computer Networks*, Nov 2004.

[14] Netfilter. (2019) Netfilter. [Online]. Available: https://www.netfilter.org/

[15] J. Mirkovic, S. Dietrich, D. Dittrich, and P. Reiher, *Internet Denial of Service: Attack and Defense Mechanisms (Radia Perlman Computer Networking and Security)*, 2004.

[16] J. Li, M. Sung, J. Xu, and L. Li, "Large-scale ip traceback in high-speed internet: practical techniques and theoretical foundation," in *IEEE Symposium on Security and Privacy, 2004. Proceedings. 2004*, May 2004, pp. 115–129.

[17] F. Seidel, K. Krentz, and C. Meinel, "Deep en-route filtering of constrained application protocol (coap) messages on 6lowpan border routers," in *2019 IEEE 5th World Forum on Internet of Things (WF-IoT)*, April 2019, pp. 201–206.

[18] X. Qie, R. Pang, and L. Peterson, "Defensive programming: Using an annotation toolkit to build dos-resistant software," in *Proceedings of the 5th Symposium on Operating Systems Design and implementation*, ser. OSDI '02. Berkeley, CA, USA: USENIX Association, 2002, pp. 45–60.

[19] N. Bhatti and R. Friedrich, "Web server support for tiered services," *IEEE Network*, vol. 13, no. 5, pp. 64–71, Sep 1999.

[20] Xiaofeng Wang and M. K. Reiter, "Defending against denial-of-service attacks with puzzle auctions," in *2003 Symposium on Security and Privacy, 2003.*, May 2003, pp. 78–92.

[21] R. J. Anderson, *Security Engineering: A Guide to Building Dependable Distributed Systems*. Wiley Publishing Inc., 2010.

[22] K. P. Birman, *Guide to Reliable Distributed Systems. Building High-Assurance Applications and Cloud-Hosted Services*. springer, 2012.

[23] Zolertia. (2019) Zolertia firefly. [Online]. Available: https://zolertia.io/product/firefly/

[24] S. Chakrabarti, G. Montenegro, and J. Woodyatt, "Ipv6 over low-power wireless personal area network (6lowpan) esc dispatch code points and guidelines," Internet Requests for Comments, RFC 8066, February 2017. [Online]. Available: https://tools.ietf.org/html/rfc8066