Received: 2 May 2016 | Revised: 9 March 2017 | Accepted: 8 May 2017

DOI: 10.1002/spe.2510

SHORT COMMUNICATION

WILEY

An experimental study on the applicability of SYN cookies to networked constrained devices

Juan Jose Echevarria[1,2] | Pablo Garaizar[1,2] | Jon Legarda[1,2]

[1]DeustoTech—Fundación Deusto, Avda. Universidades, 24, 48007, Bilbao, Spain

[2]Facultad Ingeniería, Universidad de Deusto Avda. Universidades, 24, 48007, Bilbao, Spain

Correspondence
Juan Jose Echevarria, DeustoTech—Fundación Deusto, Avda. Universidades, 24, 48007, Bilbao, Spain.
Email:juanjose.echevarria@deusto.es

Summary

The Internet protocol suite is increasingly used on devices with constrained resources that operate as both clients and servers within the Internet of Things paradigm. However, these devices usually apply few—if any—security measures. Therefore, they are vulnerable to network attacks, particularly to denial of service attacks. The well-known SYN flood attack works by filling up the connection queue with fake SYN requests. When the queue is full, new connections cannot be opened until some entries are removed after a time-out. Class 2 constrained devices—according to the RFC 7228—are highly vulnerable to this attack because of their limited available memory, even in low-rate attacks. This paper analyses and compares in a class 2 constrained device the performance of 2 commonly used defence mechanisms (ie, recycle half-open connections and SYN cookies) during a low-rate SYN flood. We first review 2 SYN cookies implementations (ie, Linux and FreeBSD) and compare them with a hybrid approach in a class 2 device. Finally, experimental results prove that the proposed SYN cookies implementation is more effective than recycling the oldest half-open connections.

KEYWORDS

constrained devices, denial of service, Internet of Things, network throughput, SYN cookies

1 | INTRODUCTION

Denial of service (DoS) attacks have targeted communication networks for decades. The exponential growth in the number of connected devices makes this threat more worrisome.[1] Internet of Things (IoT) devices are particularly vulnerable targets because they are usually resource constrained, apply few—if any—security measures,[2] and are physically more accessible, making them easier targets to compromise.[3]

RFC 7228 defines 3 classes of resource-constrained devices regarding to their limitations in the maximum code (ROM/Flash) and state (RAM) sizes.[4]

- Class 0 devices are very constrained sensor-like devices with data/code sizes below 10/100 kB. Commonly, these constrained devices require the help of a gateway to communicate with the Internet. An example of a class 0 device is WISP,[5] a programmable passive RFID tag with integrated sensors.
- Class 1 devices are slightly less constrained than class 0; they have about 10/100 kB of data/code sizes. They do not require the help of another network device to communicate with the Internet. They use minimal IP stacks and usually support protocols specifically designed for constrained devices. Most wireless sensor nodes fall into this category (eg, TelosB mote).

- Class 2 devices are far less constrained; they have approximately 50/250 kB of data/code sizes. These devices can be directly connected to the Internet as they can partially support the protocol stack used on general purpose computers. Most 32-bit microcontrollers fall into this category (eg, mbed NXP LPC1768).

Understanding common attacks against standard Internet protocols is the first step towards developing suitable security solutions.[6] A SYN flood is a layer 4 DoS attack that targets a server's Transmission Control Protocol (TCP) connection queue capacity. The attacker sends a succession of SYN requests, usually with a spoofed IP address. Those SYN requests remain in the queue until the connection is completed or the request times out. Since the SYN requests sent by the attacker are not completed, the queue becomes full and other users cannot establish new connections.[7] The memory space allocated for the connection queue is usually very small in a constrained device. Therefore, a low-rate SYN flood or even a sudden increase in incoming traffic (unintentional DoS from legitimate traffic) could disrupt a constrained device acting as a server.[8]

Several studies highlight the plausibility of DoS attacks on networks where these constrained devices may operate (eg, ad hoc networks,[9] IoT,[10] or cyber-physical systems[11]), and SYN floods are no exception.[12,13] However, the unique features of these devices have traditionally led to the development of ad hoc security solutions.[14]

One of the working hypotheses on security, privacy, and resilience of the W3C Web of Things Interest Group is that "there are existing patterns, (standard) protocols, mechanisms, components that can be reused (with or without adaptation)."[15] This is the goal of the present research. After reviewing common mitigation techniques against SYN floods (RFC 4987), we evaluate mitigation techniques in a class 2 device that can be used when the connection queue is full.[16]

The first mitigation technique is recycling the oldest half-open connection. Once the entire connection queue is full, an incoming SYN overwrites the oldest connection in the SYN-RECEIVED state. As an alternative to discarding requests, DJ Bernstein proposed SYN cookies.[17] Once the connection queue is full, an incoming request is not stored until a valid ACK is received.[18] When a SYN request arrives, the server replies with a SYN-ACK containing a SYN cookie as its Initial Sequence Number (ISN). The cookie is a cryptographic hash of some header fields from the original SYN request. Since TCP requires the client to send back that ISN on the next ACK, the server will be able to verify the cookie and, consequently, create a full connection using the encoded information.

The SYN cookies are now a standard feature of Linux and FreeBSD operating systems.[19] The purpose of reviewing these implementations is to assess their suitability for class 2 constrained devices. From that review, we realized that it is possible to address their main issues through a hybrid implementation. This implementation will be used to determine which mechanism—SYN cookies or recycling half-open connections— shows less throughput degradation during a low-rate SYN flood.

2 | PORTING SYN COOKIES TO A CLASS 2 DEVICE

Considering SYN cookies are only validated locally, their format and generation procedures vary among implementations. Despite there have been some contributions in this area,[20,21] we will only focus on the 2 implementations available in current desktop operating systems.

2.1 | Review of the SYN cookies implementations

According to Zuquete,[20] SYN cookies should comply with 4 requirements:

- Cookies must fit in the space defined for the ISN field of a TCP header (32 bits).
- Cookies should respect the TCP recommendations for ISN values being monotonically increasing over time.
- Cookies must be unpredictable by attackers.
- Cookies should contain some TCP options sent by clients in SYN packets.

Next sections show the algorithms to generate and validate cookies for both Linux and FreeBSD kernels.

2.1.1 | Linux

Linux 2.1.44 was the first version where SYN cookies were available. The original implementation encoded the connection state and a timestamp in the upper 8 bits, whereas the lower 24 bits were left for the cryptographic hash.[22] Recent kernels add another server-selected secret to make guessing more difficult. Table 1 shows the parameters of the Linux implementation. Next, we show the algorithms for generating and validating SYN cookies in the Linux 4.8 kernel.

TABLE 1 Parameters of the Linux implementation

Parameter	Meaning
K_1, K_2	Secret keys
IP_s, IP_d	Source and destination IP addresses
$Port_s$, $Port_d$	Source and destination ports
ISN_s, ISN_d	Source and destination initial sequence numbers
ACK	Acknowledgement number
SEQ	Sequence number
MSS	2 bit index of the client's Maximum Segment Size
count	32 bit minute counter
hash()	32 bit cryptographic hash (SHA-1)

Cookie generation:

$$H_1 = hash(K_1, IP_s, IP_d, Port_s, Port_d) \tag{1}$$

$$H_2 = hash(K_2, count, IP_s, IP_d, Port_s, Port_d) \tag{2}$$

$$ISN_d = H_1 + ISN_s + (count * 2^{24}) + (H_2 + MSS)\ mod\ 2^{24} \tag{3}$$

Cookie validation:

$$ISN_d = ACK - 1 \tag{4}$$

$$ISN_s = SEQ - 1 \tag{5}$$

$$count_{cookie} = (ISN_d - H_1 - ISN_s)/2^{24} \tag{6}$$

$$MSS_{cookie} = (ISN_d - H_1 - ISN_s)\ mod\ 2^{24} - H_2\ mod\ 2^{24} \tag{7}$$

As we can see, there are 2 integrity controls. If either of them fails, the cookie is rejected. The first one checks the age of the cookie. Linux SYN cookies use a monotonic timer that updates every minute. A cookie is only valid if the difference between the current counter value and the encoded one is less than 2 (MAX_SYNCOOKIE_AGE). Thus, a sent cookie is valid for a maximum of 2 minutes. The second integrity control evaluates whether the value of the MSS is within the 2 bit range (0-3). If the value is out of range, the connection is dropped. If the cookie meets both integrity controls, it is considered valid, and the connection can be accepted.

From an attacker perspective, there are 2 valid counter values and 4 possible MSS values. Therefore, attackers can skip guessing some bits when servers allow multiple MSS and recent time values.[23] In particular, the average number of packets required to successfully spoof a connection is reduced to $2^{32}/8$ during a SYN flood.

Listing 1 Relevant code for SYN cookie validity

```
#define MAX_SYNCOOKIE_AGE 2
#define TCP_SYNCOOKIE_PERIOD (60 * HZ)
#define TCP_SYNCOOKIE_VALID (MAX_SYNCOOKIE_AGE * TCP_SYNCOOKIE_PERIOD)
...
if (tcp_synq_no_recent_overflow(sk))
 goto out;
...
static inline bool tcp_synq_no_recent_overflow(const struct sock *sk)
{
 unsigned long last_overflow = tcp_sk(sk)->rx_opt.ts_recent_stamp;
 return time_after(jiffies, last_overflow + TCP_SYNCOOKIE_VALID);
}
```

On the other hand, we have found a way to reuse cookies to run a one-packet connection establishment. When the kernel receives an ACK packet, it first checks if there has been a recent queue overflow on the listening socket (tcp_synq_no_recent_overflow). The kernel does not want to accept SYN cookies if the queue has not been filled recently. However, if the kernel sends SYN cookies and then exits this mode, it will still accept cookies if they are received up to 2 minutes after the end of the SYN flood episode (TCP_SYNCOOKIE_VALID). We can see this in the code shown in Listing 1. Note that the routine time_after(a,b) returns true if the time *a* is after time *b*.

If the connection is closed on the server's end, the same combination of source address and port will be available again. The complete removal of state makes it impossible to detect replays, and thus, an ACK with a previous cookie will be assumed as a valid third packet of a TCP connection.

In this scenario, SYN cookies allow a client not to wait for a SYN-ACK most of the time. With the limited connection queue of constrained devices, a client could intentionally send the required number of packets to fill the queue (eg, with a spoofed IP address). Then, open a connection, collect the cookie, and reuse it to speed up the opening of successive TCP connections for 2 minutes. This greedy behaviour would have the same result as TCP Fast Open[24] but without the awareness of the server.

2.1.2 | FreeBSD

FreeBSD uses a SYN cache[25] to store a minimal information of the request. When the connection establishment is done, the system creates a full-state connection with the information stored in the SYN cache entry, and then this entry is released. The SYN cookies are used when the cache is full. Current implementation adds more TCP options and uses SipHash[26] as the cryptographic hash. This new implementation was merged into FreeBSD-10. Table 2 shows the parameters of the FreeBSD implementation. Next, we show the algorithms for generating and validating SYN cookies in the FreeBSD 11.0 kernel (revision 308048).

Cookie generation:

$$h = hash(K_P, IP_s, IP_d, Port_s, Port_d, ISN_s, options) \tag{8}$$

$$ISN_d = h \ \& \sim 0xff \tag{9}$$

$$ISN_d| = options \oplus (h \gg 24) \tag{10}$$

Cookie validation:

$$ISN_d = ACK - 1 \tag{11}$$

$$ISN_s = SEQ - 1 \tag{12}$$

$$options = (ISN_d \ \& \ 0xff) \oplus (ISN_d \gg 24) \tag{13}$$

$$h = hash(K_P, IP_s, IP_d, Port_s, Port_d, ISN_s, options) \tag{14}$$

$$(ISN_d \ \& \sim 0xff) == (h \ \& \sim 0xff) \tag{15}$$

TABLE 2 Parameters of the FreeBSD implementation

Parameter	Meaning
IP_s, IP_d	Source and destination IP addresses
$Port_s$, $Port_d$	Source and destination ports
ISN_s, ISN_d	Source and destination initial sequence numbers
ACK	Acknowledgement number
SEQ	Sequence number
MMM	3 bit index of the client's MSS
WWW	3 bit index of the client's window scale
S	1 bit that sets if selective acknowledgement is allowed
P	1 bit that sets which of the two secrets is in use
options	8 bits (WWWMMMSP)
K_P	Secret key in use
hash()	32 bit cryptographic hash (SipHash)

The lookup routine first retrieves the key used to produce the cookie. Instead of using a counter encoded in the cookie, the FreeBSD algorithm relies on 2 rotating keys to produce a cookie. Every 15 seconds one of the keys is updated with a new random value, and the cookie encodes which key has been used. A cookie is only valid if the received ACK matches the recomputed hash. As only 24 bits are used for the cryptographic hash, the average number of packets required to successfully spoof a connection can be reduced to 2^{24}. However, unlike Linux, this can happen even when the system is not under a SYN flood.

If enabled, SYN cookies are generated for every received SYN packet.[27] When the kernel receives an ACK packet for a listening socket, it determines if the connection is in the SYN cache. If there is no SYN cache entry, the kernel checks if the ACK is a returning SYN cookie. As the kernel does not check whether there has been a recent overflow before accepting cookies, it might be possible for an attacker to ACK flood a server in an attempt to create a connection. This may also provide a way to bypass firewalls that filter incoming packets with the SYN bit set. Furthermore, the cost of validating cookies may be substantial compared to processing regular ACK packets.

Regarding the reuse of cookies, they are generated for every received SYN and the validation routine does not apply any condition to stop accepting cookies. Therefore, a client could collect and replay cookies to effectively run a one-packet connection establishment. Despite the cookies are valid for a maximum of 30 seconds (ie, 4 times less than in Linux), there is no need to overflow the SYN cache for reusing them.

2.2 | Hybrid implementation

Firstly, we believe that the expiration time of both Linux and FreeBSD implementations is excessive for regular connection establishments. The RTT (round trip time) between a SYN-ACK and the corresponding client's ACK is usually less than 500 milliseconds for more than 90% of connections.[28] Experimental tests in the Internet prove the aforementioned, with a mean RTT between the SYN-ACK and the returning ACK of 145 milliseconds.[29]

On the other hand, FreeBSD not only does not check whether there has been a recent overflow in the listening socket but also requires a periodic routine to update the cryptographic keys. Both design decisions are not well suited for a constrained device. Considering this, we will use the Linux implementation as the foundation for our hybrid proposal. Our goal is to reduce the expiration time of cookies and improve performance by using the same cryptographic hash as in FreeBSD.

As we explained previously, Linux SYN cookies expire by encoding a counter that is incremented every minute. When generating a SYN cookie, the system gathers the number of ticks that have occurred since the kernel booted and then converts that value to minutes. To reduce the expiration time of cookies, we propose to do a conversion to seconds instead of minutes.

Despite that the size of the minute timer is 32 bits, the way it is encoded in the cookie (upper 8 bits) makes it roll over every 256 minutes. This means that if the rest of the parameters are the same, the value of a cookie generated every 256 minutes would be the same. If we encode the counter of seconds of our proposal in the same way, it will roll over every 256 seconds, which is a period too short to be considered a robust SYN cookies implementation. Therefore, we will use 14 bits for this counter so it rolls over similarly (ie, 16 384 seconds = 273 minutes).

Table 3 shows the parameters of our hybrid proposal. Next, we present the generation and validation algorithms.

Cookie generation:

$$H_1 = hash(K_1, IP_s, IP_d, Port_s, Port_d) \tag{16}$$

$$H_2 = hash(K_2, count, IP_s, IP_d, Port_s, Port_d) \tag{17}$$

$$ISN_d = H_1 + ISN_s + (count * 2^{18}) + (H_2 + MSS)\ mod\ 2^{18} \tag{18}$$

TABLE 3 Parameters of the hybrid implementation

Parameter	Meaning
K_1, K_2	Secret keys
IP_s, IP_d	Source and destination IP addresses
$Port_s$, $Port_d$	Source and destination ports
ISN_s, ISN_d	Source and destination initial sequence numbers
ACK	Acknowledgement number
SEQ	Sequence number
MSS	2 bit index of the client's MSS value
count	32 bit seconds counter
hash()	32 bit cryptographic hash (SipHash)

Cookie validation:

$$ISN_d = ACK - 1 \tag{19}$$

$$ISN_s = SEQ - 1 \tag{20}$$

$$count_{cookie} = (ISN_d - H_1 - ISN_s)/2^{18} \tag{21}$$

$$MSS_{cookie} = (ISN_d - H_1 - ISN_s)\ mod\ 2^{18} - H_2\ mod\ 2^{18} \tag{22}$$

In our hybrid proposal, we apply the same integrity controls used in the Linux implementation. A cookie is only valid if the difference between the current counter value and the encoded one is less than 2 (MAX_SYNCOOKIE_AGE), ie, it accepts cookies aged up to 2 seconds. Similarly, the server would also accept 8 valid combinations at any given time, as it also accepts 4 MSS values. Finally, the TCP_SYNCOOKIE_VALID variable is also reduced to 2 seconds for consistency. Nevertheless, these values can be slightly increased to perform correctly on networks with higher RTT. In short, a cookie generated with our hybrid algorithm is valid only for time periods on the order of a few seconds, which makes reuse of cookies much more difficult.

2.3 | Performance evaluation

In this section, we show a performance evaluation of the 3 SYN cookies algorithms (Linux, FreeBSD, and hybrid) in a class 2 constrained device: the LPC1768. To connect this device to a local intranet, we used the 1.4.0 version of Lightweight IP (lwIP),[30] an open source TCP/IP networking stack designed for devices with limited resources.

The lwIP uses a single connection queue. The size of this queue is determined by the MEMP_NUM_TCP_PCB argument. When lwIP receives an IP packet, the ip_input routine is called (see Figure 1). This routine does the basic checks of the IP header and sends the packet to the upper layer protocol input routine. In the case of TCP packets, the processing is done in tcp_input. When a packet arrives for a listening socket, the tcp_listen_input routine is called. For a SYN packet, the routine tries to create a new entry in the queue, then processes the header fields, and finally replies with a SYN-ACK. If there is no room in the queue for a new entry, the connection is silently dropped. Finally, if an ACK packet is received for a listening socket, the routine replies with a RST packet.

This study compares the performance of the different SYN cookies generation algorithms during a low-rate SYN flood. For that aim, we have modified the behaviour of the tcp_listen_input routine so all requests are handled with SYN cookies. When a SYN packet is received, the routine computes the SYN cookie and sends back the SYN-ACK packet. When an ACK is received, the routine checks if the ACK is a returning SYN cookie. If so, the connection is added to the queue and becomes eligible for handover to the application; otherwise, the connection is dropped.

We choose the network throughput as the performance metric. In communication networks, network throughput is one of the key performance metrics[31] and measures the rate of successful data delivery in a given time frame, eg, megabits per second (Mbps). In the procedure followed to conduct this study, the client side will send TCP packets to the server as fast as possible for a period of 30 seconds. The server will then acknowledge those packets as fast as possible. Only the throughput from the client to the server is measured.

We collect 30 samples for each SYN cookies implementation, as that sample size[32] would provide enough observations to estimate the implementation that degrades the network throughput less. To find out how effective these implementations are, we set up a test environment to simulate a TCP server configuration. The test environment consists of the following hosts:

- Attacker: acts as the source of the SYN flood. The host is a 3.4 GHz Intel Pentium 4 with 3GB RAM running Ubuntu 12.04.
- Server (victim): runs a TCP server and is the destination of the SYN flood. The host is the class 2 device (LPC1768).
- Client: accesses the TCP service from the server. The host is a 2.67 GHz Intel Core i5-560 M with 6GB DDR3 RAM running Ubuntu 14.04.

TABLE 4 Throughput in Mbps

	Mean	Std Dev
No attack	33.72	0.045
FreeBSD cookie	31.94	0.094
Linux cookie	31.67	0.073
Hybrid cookie	31.91	0.081

A standard 100Base-T switch provides the required network connectivity between all hosts. To generate the SYN flood attack, we use the *hping* tool. To measure the throughput, we use *Iperf* to create a TCP data stream between the client and the server. The output contains a time-stamped report of the amount of data transferred and the throughput measured. The throughput results are collected for further analysis.

Finally, we must choose the SYN flood rate. We can classify SYN attacks into 2 categories: high- and low-rate attacks. We decided to set the rate of SYN packets to 5000μs (200 packets/second), an attack that could be qualified as low-rate on desktop environments, but for a class 2 device and its limited connection queue, we believe it is a significant rate to determine the best SYN cookies implementation. Table 4 shows the results of this study.

As we can see, there is a throughput degradation when the device is under a low-rate attack. Results show that the algorithms used for the cookie computation have different means. To evaluate if the difference in means is significant, we use the unpaired *t* test and the effect size (Cohen d).

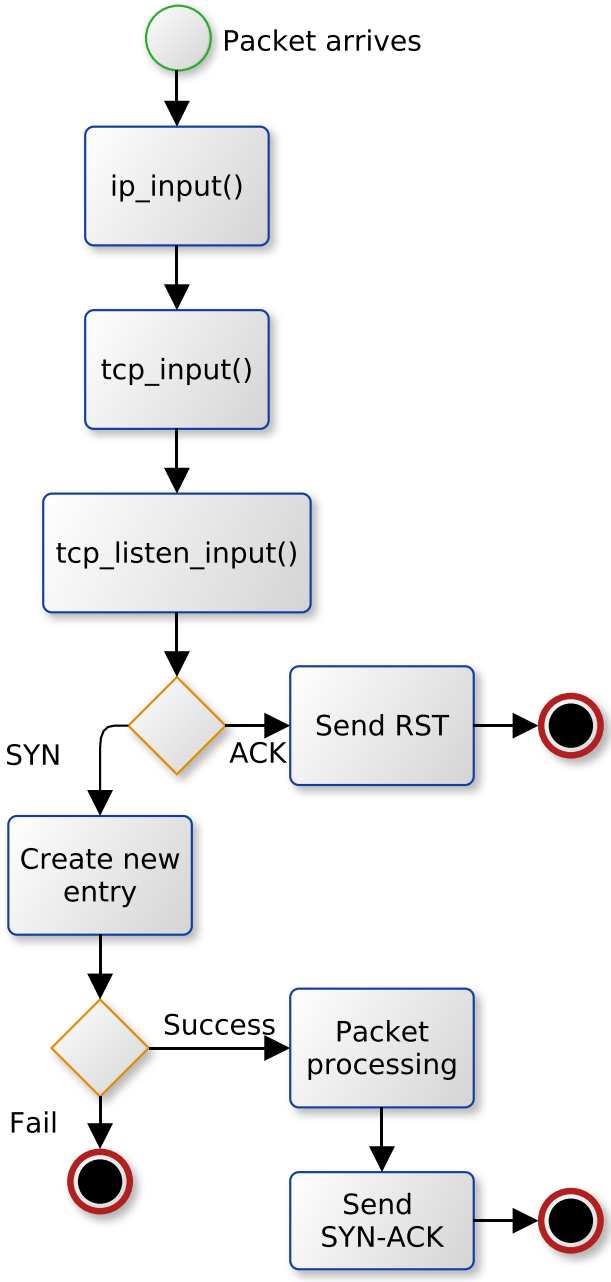

FIGURE 1 Transmission Control Protocol (TCP) processing for a listening socket [Colour figure can be viewed at wileyonlinelibrary.com]

Among SYN cookies implementations (N = 30), there was a statistically significant difference between Linux (M = 31.67, SD = 0.073) and FreeBSD (M = 31.94, SD = 0.094), $t(58) = 12.4256$, $P \leq .05$, CI.95 from −0.31350 to −0.22650. Therefore, we reject the null hypothesis that there is no difference in network throughput between Linux and FreeBSD implementations. Furthermore, Cohen effect size value (d = 3.2) suggested a very high practical significance. In the same way, there was a statistically significant difference between Linux (M = 31.67, SD = 0.073) and our hybrid proposal (M = 31.91, SD = 0.081), $t(58) = 12.0554$, $P \leq .05$, CI.95 from −0.27985 to −0.20015. Thus, we also reject the null hypothesis that there is no difference between those implementations. Moreover, Cohen effect size value (d = 3.11) suggested a very high practical significance. On the other hand there was no statistically significant difference between FreeBSD (M = 31.94, SD = 0.094) and our hybrid proposal (M = 31.91, SD = 0.081), $t(58) = 1.3242$, $P \geqslant .05$, CI.95 from −0.01535 to 0.07535. Therefore, we fail to reject the null hypothesis that there is no difference in network throughput between FreeBSD and our hybrid implementation. Furthermore, Cohen effect size (d = 0.34) suggested a low to moderate practical significance.

The performance evaluation proves that the Linux implementation might be more suited for high-end devices due to the CPU time involved in computing 2 SHA-1 hashes per connection request. On the other hand, FreeBSD and our hybrid implementation seem more suited for low-end devices (eg, class 2 devices) because they use SipHash, a performance-optimized cryptographic hash. In this case, the cryptographic hash is the main source for the statistical variation.

Hence our hybrid solution makes the reuse of cookies more difficult, while using 2 SipHash computations is not statistically different from using one (FreeBSD). We will use this hybrid SYN cookies implementation for the analysis presented in the next section.

3 | SYN COOKIES VS RECYCLING CONNECTIONS

Once we have found that our hybrid solution performs properly on a constrained device, we want to study which mitigation mechanism—SYN cookies or recycling half-open connections—works best in a class 2 constrained device. The experimental setup is the same used in the study shown in the previous section. Similarly, the network throughput (in Mbps) is the dependant variable of this study. We compare 5 different configurations in our class 2 device:

- Size_2: The size of the connection queue is 2, and if it fills up, the oldest half-open entry is replaced.
- Size_4: The size of the connection queue is 4, and if it fills up, the oldest half-open entry is replaced.
- Size_8: The size of the connection queue is 8, and if it fills up, the oldest half-open entry is replaced.
- Size_16: The size of the connection queue is 16, and if it fills up, the oldest half-open entry is replaced.
- Cookies: Instead of replacing the oldest entries in the connection queue, incoming SYN requests are processed with cookies.

Table 5 shows the descriptive statistics for the dependant variable.

A one-way between subjects ANOVA was conducted to compare the effect of the configuration on the network throughput. On the basis of the results, there was a statistically significant difference at the $P \leq .05$ level for the 5 configurations [$F(4,145) = 938.966$, $P = .000$]. Post hoc comparisons using the Bonferroni test showed that all means were significantly different ($P = .000$). Taken together, these results suggest that the queue size really does have an effect on the network throughput. Specifically, our results suggest that the network throughput is higher as the connection queue gets smaller. Nevertheless, no matter the queue size, results show that recycling half-open connections is less effective than using SYN cookies.

Given a data structure like the connection queue, the search time is O(n) if not sorted, and O(log(n)) if sorted. With a small queue size, the results are closer to the SYN cookies. However, as the size increases, the difference in means is larger. This

TABLE 5 Descriptive statistics

	Configuration				
	Size_2	**Size_4**	**Size_8**	**Size_16**	**Cookies**
N	30	30	30	30	30
Mean	31.7557	31.6113	31.4400	31.3297	31.9107
Std Deviation	0.03126	0.06307	0.03778	0.03891	0.02924
Std Error	0.00571	0.01151	0.00690	0.00710	0.00534
Lower bound	31.7440	31.5878	31.4259	31.3151	31.8997
Upper bound	31.7673	31.6349	31.4541	31.3442	31.9216
Minimum	31.68	31.50	31.38	31.25	31.86
Maximum	31.79	31.70	31.51	31.39	31.97

proves that the linear search is the main source for the throughput degradation. On the other hand, a server that uses SYN cookies does not have to perform a linear search, it simply computes cookies and replies with SYN-ACKs. The cookie computation is constant (O(1)) as long as the input parameters have the same size.

4 | CONCLUSION

Any Internet-connected device is susceptible to DoS attacks, but constrained devices are particularly vulnerable because of their limited computational resources. Devices listed in RFC 7228 should also apply security measures when the connection queue becomes full of fake SYNs; otherwise, the service provided would be quickly disrupted.

To the best of our knowledge, this is the first work that applies SYN flood defence mechanisms listed in the RFC 4987 to resource-constrained devices. The experiments were designed to measure the degradation of the network throughput regarding how SYN packets were processed at the class 2 device.

We found that the difference between the 5 configurations is statistically significant. This means SYN cookies are more effective under a low-rate SYN flood than recycling the oldest half-open connections. Besides, recycling entries in such a small queue may prevent legitimate connections if those establishments take more time than the attack SYNs to fill the queue.

Finally, we prove that the development of any piece of software or security measure for constrained devices should be carefully designed and tested, since it ultimately affects network throughput.

ACKNOWLEDGEMENTS

This work has been supported in part by a predoctoral fellowship from the Department for Education, Language Policy, and Culture of the Basque Government.

REFERENCES

1. Masdari M, Jalali M. A survey and taxonomy of dos attacks in cloud computing. *Secur Commun Networks*. 2016;9(16):3724-3751. SCN-15-0746.R1.
2. Ar A, Oktu SF, Yalmn SB. Internet-of-things security: denial of service attacks. In *2015 23nd Signal Processing and Communications Applications Conference (SIU)*. Malatya, Turkey: IEEE; 2015: 903-906.
3. Lopez J, Zhou J. *Wireless Sensor Network Security*. Amsterdam, The Netherlands, The Netherlands: IOS Pres; 2008.
4. Bormann C, Ersue M, Kernen A. Terminology for constrained-node networks. RFC 7228, 2015.
5. Sample AP, Yeager DJ, Powledge PS, Mamishev AV, Smith JR. Design of an RFID-based battery-free programmable sensing platform. *IEEE Trans Instrum Meas*. 2008;57(11):2608-2615.
6. Geetha K, Sreenath N. Syn flooding attack: identification and analysis. In: *Information Communication and Embedded Systems (ICICES), 2014 International Conference on*. Chennai, India: IEEE; 2014: 1-7.
7. Schuba CL, Krsul IV, Kuhn MG, Spafford EH, Sundaram A, Zamboni D. Analysis of a denial of service attack on tcp. In: *Security and Privacy, 1997. Proceedings., 1997 IEEE Symposium on*. Oakland, CA, USA: IEE; 1997: 208-223.
8. Kavisankar L, Chellappan C. *T-RAP: (TCP Reply Acknowledgement Packet) a Resilient Filtering Model for DDoS Attack with Spoofed IP Address, pages 138-148*. Berlin, Heidelberg; Springer Berlin Heidelberg; 2011.
9. Aad I, Hubaux JP, Knightly EW. Impact of denial of service attacks on ad hoc networks. *IEEE/ACM Trans Networking*. 2008;16(4):791-802.
10. Pa YMP, Suzuki S, Yoshioka K, Matsumoto T, Kasama T, Rossow C. Iotpot: analysing the rise of iot compromises. In: *9th USENIX Workshop on Offensive Technologies (WOOT 15)*, Washington, D.C. USENIX Association; 2015: 9-9
11. Nur AY, Tozal ME. Defending cyber-physical systems against dos attacks. In: *2016 IEEE International Conference on Smart Computing (SMARTCOMP)*, St. Louis, MO, USA: IEEE; 2016: 1-3.
12. Chen B, Pattanaik N, Goulart A, Butler-purry KL, Kundur D. Implementing attacks for modbus/tcp protocol in a real-time cyber physical system test bed. In: *2015 IEEE International Workshop Technical Committee on Communications Quality and Reliability (CQR)*. Charleston, SC, USA: IEEE; 2015: 1-6.
13. Hahn A, Ashok A, Sridhar S, Govindarasu M. Cyber-physical security testbeds: Architecture, application, and evaluation for smart grid. *IEEE Trans Smart Grid*. 2013;4(2):847-855.
14. Misra S, Maheswaran M, Hashmi S. *Securing the Internet of Things*. Springer International Publishing: Cham, 2017; 39-51.
15. Security, privacy and resilience, 2015. https://www.w3.org/WoT/IG/wiki/Security,_Privacy_and_Resilience
16. Eddy W. TCP SYN Flooding Attacks and Common Mitigations. RFC 4987, 2015.
17. Bernstein DJ. Syn cookies. http://cr.yp.to/syncookies.html
18. Beitollahi H, Deconinck G. Analyzing well-known countermeasures against distributed denial of service attacks. *Comput Commun*. 2012;35(11):1312-1332.
19. Mahimkar A, Dange J, Shmatikov V, Vin H, Zhang Y. Dfence: Transparent network-based denial of service mitigation. In: *Proceedings of the 4th USENIX Conference on Networked Systems Design & Implementation*, NSDI'07. Berkeley, CA, USA: USENIX Association; 2007: 24-24.

20. Zuquete A. Improving the functionality of syn cookies. In: *Proceedings of the IFIP TC6/TC11 Sixth Joint Working Conference on Communications and Multimedia Security: Advanced Communications and Multimedia Security* Kluwer, B.V.: Deventer, The Netherlands, The Netherlands; 2002: 57-77.
21. Hang B, Hu R. A novel SYN cookie method for tcp layer ddos attack. In: *2009 International Conference on Future Biomedical Information Engineering (FBIE)*. Sanya, China: IEEE; 2009: 445-448.
22. Smith C, Matrawy A. Comparison of operating system implementations of SYN flood defenses (cookies). In: *Communications, 2008 24th Biennial Symposium on*. Kingston, ON, Canada: IEEE; 2008: 243-246.
23. Syn cookies. https://cr.yp.to/syncookies.html. (Accessed on October 15, 2016)
24. Radhakrishnan S, Cheng Y, Chu J, Jain A, Raghavan B. Tcp fast open. In: *Proceedings of the 7th International Conference on Emerging Networking Experiments and Technologies (CONEXT)*. Tokyo, Japan: ACM; 2011: 21:1(21);12
25. Lemon J. Resisting SYN flood dos attacks with a SYN cache. In: *Proceedings of the BSD Conference 2002 on BSD Conference*, BSDC'02 Berkeley, CA, USA: USENIX Association; 2002: 10-10.
26. Aumasson J-P, Bernstein DJ. Siphash: a fast short-input prf. In: *Progress in cryptology - indocrypt 2012*, Galbraith S, Nandi M (eds)., Lecture Notes in Computer Science, vol. 7668 Berlin Heidelberg; Springer; 2012: 489-508.
27. McKusick MK, Neville-Neil G, Watson RNM. *The Design and Implementation of the Freebsd Operating System*. 2nd edn. Upper Saddle River, NJ, USA:Addison-Wesley Professional;2014.
28. Korczynski M, Janowski L, Duda A. An accurate sampling scheme for detecting SYN flooding attacks and portscans. In: *Communications (ICC), 2011 IEEE International Conference on*. Kyoto, Japan: IEEE; 2011:1-5.
29. Sessini P, Mahanti A. Observations on round-trip times of TCP connections. In: *International Symposium on Performance Evaluation of Computer and Telecommunication Systems*. Calgary, Canada: Society for Modeling and Simulation International; 2006:347-354.
30. Dunkels A. Design and implementation of the lwip, 2001. http://www.nongnu.org/lwip. (Accessed on October 15, 2016)
31. Darst C, Ramanathan S. Measurement and management of internet services. In: *Integrated Network Management VI. Distributed Management for the Networked Millennium. Proceedings of the Sixth IFIP/IEEE International Symposium on Integrated Network Management. (Cat. No.99ex302)*, Boston, MA, USA: IEEE; 1999: 125-140.
32. Corder GW, Foreman DI. *Nonparametric Statistics for Non-Statisticians: A Step-by-Step Approach*. New Jersey: Wiley; 2009.

How to cite this article: Echevarria JJ, Garaizar P, Legarda, J. An experimental study on the applicability of SYN cookies to networked constrained devices. *Softw Pract Exper*. 2017;1-10. https://doi.org/10.1002/spe.2510