SAFETY: Early Detection and Mitigation of TCP SYN Flood Utilizing Entropy in SDN

Prashant Kumar, Meenakshi Tripathi, *Member, IEEE*, Ajay Nehra, Mauro Conti, *Senior Member, IEEE*, and Chhagan Lal

***Abstract*—Software defined networking (SDN) is an emerging network paradigm which emphasizes the separation of the control plane from the data plane. This decoupling provides several advantages such as flexibility, programmability, and centralized control. However, SDN also introduces new vulnerabilities due to the required communication between data plane and control plane. Examples of threats that leverage such vulnerabilities are the control plane saturation and switch buffer overflow attacks. These attacks can be launched by flooding the TCP SYN packets from data plane (i.e., switches) to the control plane. This paper presents SAFETY, a novel solution for the early detection and mitigation of TCP SYN flooding. SAFETY harnesses the programming and wide visibility approach of SDN with entropy method to determine the randomness of the flow data. The entropy information includes destination IP and few attributes of TCP flags. To show the feasibility and effectiveness of SAFETY, we implement it as an extension module in *Floodlight controller* and evaluate it under different conditional scenarios. We run a thorough evaluation of our implementation through extensive emulation via *Mininet*. The experimental results show that when compared to the state-of-the-art, SAFETY brings a significant improvement (13%) regarding processing delay experienced by a legitimate node. Other parameters such as CPU utilization at the controller and attack detection time are also examined and shows improvement in various scenarios.**

***Index Terms*—Denial-of-service (DDoS), software-defined networking, security, entropy, TCP SYN-flooding.**

Manuscript received October 6, 2017; revised March 22, 2018 and June 22, 2018; accepted July 22, 2018. Date of publication July 31, 2018; date of current version December 10, 2018. The work is partially supported by DEiTY Government of India project, ISEA-II in the Department of Computer Science and Engineering at Malaviya National Institute of Technology, Jaipur. Mauro Conti is supported by a Marie Curie Fellowship funded by the European Commission (agreement PCIG11-GA-2012-321980). This work is also partially supported by the EU TagItSmart! Project (agreement H2020-ICT30-2015-688061), and by the projects "Physical-Layer Security for Wireless Communication", and "Content Centric Networking: Security and Privacy Issues" funded by the University of Padua. The associate editor coordinating the review of this paper and approving it for publication was S. Latre. *(Corresponding author: Chhagan Lal.)*

P. Kumar, M. Tripathi, and A. Nehra are with the Department of Computer Science and Engineering, Malaviya National Institute of Technology, Jaipur 302017, India (e-mail: prashantkr.kumar88@gmail.com; mtripathi.cse@mnit.ac.in; 2014rcp9553@mnit.ac.in).

M. Conti is with the Department of Mathematics, University of Padova, 35122 Padua, Italy (e-mail: conti@math.unipd.it).

C. Lal is with Department of Computer Science and Engineering, Manipal University Jaipur, Jaipur 303007, India, and also affiliated with University of Padova, 35122 Padua, Italy. (e-mail: chhagan.iiita@gmail.com).

Digital Object Identifier 10.1109/TNSM.2018.2861741

I. Introduction

SOFTWARE Defined Networking (SDN) is a promising network paradigm aiming at decoupling control and data planes to increase network programmability and management [1]. In SDN, the applications have dynamic and granular access to the network resources and specify traffic flow policies over the networking infrastructure. In SDN network, the application requirements are forwarded through Northbound APIs to the SDN controller. The controller is in-charge of translating applications requirements into adequate forwarding rules to be enforced over the underlying network. To this purpose, the Southbound interface allows the SDN controller to access control plane functions (e.g., reporting network status, and managing packet forwarding rules at data plane) provided by the networking devices.

The use of standardized interfaces such as Openflow [2], allows to increase interoperability among network elements, avoiding vendor lock-in issues. Despite these advantages, SDN also carries a number of security threats which are inherently present in the traditional network. One of such threat is Denial of Service (DoS). TCP SYN flooding is the most popular among all DoS attacks that exploit the TCP vulnerability at the Web server end. A Web server (or any type of service that uses TCP) depends on TCP as the underlying transport layer protocol. A Web server keeps some embryonic (half-open) connections until the corresponding final ACK arrives from the interacting clients. Keeping such connections require certain amount of memory reserved at the Web server. Attacker(s) can open a large number of such incomplete connections, thus depletes the SYN-Queue easily which will ultimately deny or delay the legitimate connections request. Therefore, any network service that binds to and listens on a TCP socket is potentially vulnerable to TCP SYN flooding attacks.

A. Motivation and Contribution

Traditional networks use end host defense measures such as SYN cookies, SYN cache [3], SYN proxy [4], and middleboxes such as firewalls and Intrusion Detection Systems (IDS) for the mitigation of TCP SYN flood attacks. SYN cookies, SYN cache, and SYN proxy, these all require a modification in their underlying TCP/IP stack implementation, while the firewall needs to be implemented in a gateway through Access Control Lists (ACL). IDS works on signature-based mechanism just like some Anti-Virus solution. If traffic traces matches with some signatures from pre-stored database, IDS

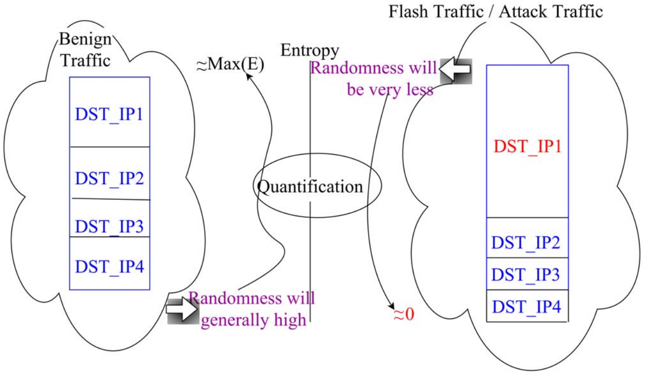


Fig. 1. Limitation of entropy on destination IP in discriminating flash crowd vs DDoS traffic.

generates an alert to the Intrusion Prevention System (IPS) for the handling that anomaly in future. All these solutions are deployed over the end-host machines. So, a distributed approach requires a global view of the network traces. The recent emergence of SDN offers an opportunity for a better anti-DDoS design due to programmability which comes from the separation of control and data plane. Since SDN provides a global view of the network traces, and continuous monitoring of these traces assists in providing a divide-conquer strategy at the source end of traffic generation.

As a statistical metric, entropy-based detection scheme has been quite prevalent in the literature of traditional as well as SDN networks [5], [6]. Entropy represents the degree of randomness associated with a variable. During DDoS, entropy distribution concerning destination IP tends to be less random. Figure 1 discusses an important observation for benign traffic, flash traffic and malicious traffic. The flash traffic is the sudden traffic from various legitimate sources to a particular destination. The flash traffic is also benign but it is voluminous concerning time. The randomness concerning a destination IP would be very high i.e., entropy $\approx max(E)$ in case of benign. However, if the concentration of a particular IP *DST_IP*1 is large, i.e., entropy ≈0 which is true in the case of flash traffic. To this end, our proposal extends the range of the chosen random variable (i.e., composite random variable), and evaluates entropy based on destination IP with the aid of TCP flag attributes.

In particular, the main contributions of this research are as follow.

- We investigate and analyze the effect of TCP SYN flood attack and presents attack manifestation in different scenarios.
- We propose and fully implement SAFETY, a novel solution that efficiently detect and mitigate SYN flood attack in SDN. SAFETY algorithm starts with entropy computations on destination IP, and it uses a set of selected TCP flags as the random variables which is then followed by the attacker identification using adaptive threshold. Additionally, we show that SAFETY also protects the network from control plane saturation attack and buffer overflow attack.
- We perform an extensive evaluation of SAFETY which shows significant improvement regarding average response time, average attack detection time, and average CPU utilization as compared to the state-of-art methods. Furthermore, we provide the source code of our proposed work on github for interested researchers in the area.

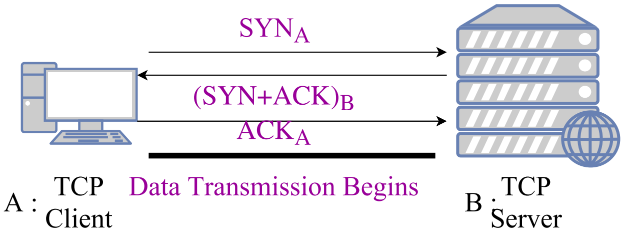


Fig. 2. Normal TCP 3-way handshake mechanism.

B. Organization

The rest of the paper is structured as follows. Section II presents the related literature survey that includes the state-of-the-art defenses for TCP SYN flooding attack. Our proposed algorithm to detect and mitigate the SYN flooding, called *SAFETY*, is described, in detail, in Section III. In Section IV, the result of simulation of SYN flood attack and its manifestation is explained along with the performance evaluation of our proposed approach. Finally, Section V concludes the paper with suggestive future enhancements.

II. Related Work

In this section, we provide an overview of the TCP three-way handshaking process and relevant literature for the mitigation of SYN flood attacks in SDN networks.

The TCP connection process starts with the TCP three-way handshake mechanism initiated by a client [7]. The process of TCP three-way handshaking in depicted in Figure 2. A generic DDoS detection approach is proposed in [8]. The approach consists of a set of steps that includes flow collection, important feature extraction, and self organized maps (SOM) classification. The features for DDoS are 6-tuples entries, i.e., average of packets per flow, average of bytes per flow, average of duration per flow, percentage of pair-flows, growth of single-flows, and growth of different Ports. The paper assumes that for each attack packet that is received from different source, there should be a new flow entry, but it is always implementation dependent. Also various topology scenarios such as near to victim or near to attacker are not discussed. Chin *et al.* [9] discuss selective packet inspection technique to detect DoS attacks. However, the paper uses a static threshold to detect the attack and it does not discuss any mitigation strategy.

Due to the wide popularity of SYN flood attacks, many solutions have been proposed in the literature to counter such attacks in SDN networks. The solutions found in the literature are either implemented on the edge switch at the data plane or on the controller at the control plane. AVANT-GUARD [10] has been proposed as a solution for the TCP SYN flood directed towards the controller. It adds a *connection migration* module and a *actuating trigger* module in the data plane of switches. The *connection migration* module acts as a proxy for the incoming TCP connections between two peers and shields the control plane from DDoS attacks. Inspired by SYN proxy technique, it will intercept the TCP connection information in a session and will forward the request to the controller only for such connections which have been established.

Downside:

- There should be an Avant-Guard agnostic controller to work with Avant-Guard extended OpenFlow switches,
- Prone to a new type of Denial of Service attack (Buffer saturation attack) at the data plane of switches [11],
- Once verified the connection establishment, the *connection migration* module have to re-initiate the handshaking procedure with the end host, thus even a legitimate connection request has to suffer a delay for the accession of services [11],
- Owing to data plane implementation, switches have to provide an extension to support this feature.

Nugraha *et al.* [12] use a sampling tool (sFlow) for the detection and mitigation of SYN flood attacks. The methodology relies on the sFlow for capturing, and it sums the traffic from each agent to send to a collector for further analysis. The authors has taken sampling interval of every 30 seconds for polling switch statistics. When the sum cumulative traffic (basically SYN requests) of all the switches exceeds a certain threshold (T), the attack would be flagged.

Downside:

- There is a need to configure sFlow as an agent in every OpenFlow switches,
- Detection accuracy largely depends on the sampling accuracy,
- Detection mechanism also affects benign requests,
- No discrimination of flash crowd (a huge surge of benign request due to some rush hour) from DDoS traffic which yields a high number of false positives,
- A static value of threshold has been chosen in this paper,
- No discussion on the mitigation part of the problem description.

Wang *et al.* [6] propose an entropy based DDoS detection mechanism implemented at the edge OF-switches. The authors extend a copy of the packet number counter of each flow table entry in the OpenFlow table called as "*RP_Local*". The entropy has been calculated by each OF edge switch based on the destination IP's which are present in their own network.

Downside:

- It requires placing the proposed solution in every OF-edge switch as part of DDoS detection,
- Every edge switch has the view its local network traffic only (thus cannot have a global scenario of network traces),
- Owing to data plane implementation, switches need to upgrade to support this feature,
- The proposed work has assumed a static threshold,
- No ground has been made to discriminate flash crowd from the attack traffic.

Dhawan *et al.* [13] propose SPHINX, which uses flow graphs to abstract network behavior. Flow graphs are incrementally built upon OpenFlow messages, i.e., FLOW_MOD, PACKET_IN, STATS_REPLY and FEATURES_REPLY. SPHINXs also gives policy language to specify and verify constraints. This verification process helps to validate network behavior over the pre-specified threshold. Additionally, the use of the abstraction of flow graphs which are approximating the actual network operations raise alerts if it identifies deviant behavior.

Downside:

- The paper does not discuss TCP SYN flooding attack.
- It uses static binding for thresholds, hence it cannot be generic.

Fichera *et al.* [14] propose **OPE**nflow-based **RE**medy **T**o **T**CP SYN-flood **A**ttacks (OPERETTA) as a solution for mitigating TCP SYN flooding. The OPERETTA detection mechanism is implemented in the _handle_PacketIn_ function, the controller proxies the incoming TCP connections and first validate the legitimacy of a source host. In OPERETTA, the data plane switches have to send every new SYN packets to the controller, and the controller (on behalf of a Web server) responds with a SYN-ACK to the client. If the client sends the ACK then controller has successfully validated the client legitimacy. After validating, it will install a forwarding flow rule in the access switch (of the host) that enables subsequent SYN for that client to reach to the server without the controller involvement. Further, with successful authentication, the controller needs to generate an RST[1] segment to send to that client, and the client have to further initiate the request once again. If the client does not send back the ACK that source MAC is put into a temporary list and its corresponding SYN counter will be incremented up to a predefined threshold. An SYN counter is used to denotes the number of concurrent half-open connections reserved for a specific node identified through a certain MAC address. This counter denotes the number of SYN request not yet ACKed for a particular host. Its threshold has been set to 10 for the OPERETTA protocol. Exceeding the threshold (showing malicious behavior), the respective client (its source MAC) is put on a blacklist for a time interval δ.

Downside:

- This algorithm needs to break the TCP end-to-end semantics,
- A significant delay in the HTTP response time even for a benign host (range from 1.00-1.20 seconds),
- Once verifying her identity, the attacker can still launch SYN flood,
- OPERETTA suffers from MAC spoofing.

Mohammadi *et al.* proposed SLICOTS [15], as an extension module in the OpenDayLight controller is a SDN-based countermeasure for TCP SYN flooding attack. First, all the TCP communications are stored on their nature of set-flag in a "*pending_list*". The pending list attaches every connection record with either $\langle$ *SYN, SYN_ACK, RST* $\rangle$. Any record on "pending_list" is counted as an illegitimate record, and if the number of illegitimate records for a specific host exceeds a predefined threshold (K), it will install a DROP rule on the edge switch of that host.

Downside:

- SLICOTS suffers from MAC spoofing,
- Statically chosen threshold of the pending connections associated with a host,

[1] A control bit in TCP header, it is used to reset the connection.

- Paper does not provide any discriminating behavior between flash crowd vs. attack traffic.

Yu *et al.* [16] presents SENSS, a victim-oriented DDoS detection and mitigation approach in which victim initiates a request to ISP for traffic query and request to take mitigation actions. However, the paper does not discuss when victim generates traffic query request, i.e., periodic or after attack detection Additionally, if an attack has already performed on a victim and it is assumed that victim will interact with ISP, it leads to delayed detection of the attack which increases bandwidth wastage. Giotis *et al.* [17] propose a sFlow-based DDoS detection and mitigation approach to handle DDoS, Worm propagation, and Portscan attacks. Information related to flows are collected at the controller which is then used to calculate the entropy to distinguish between attacks. One identified, new flow entries are generated to prevent the attack packets to reach to the target. However, to detect Flash Crowd it uses a whitelist, which is a resource intensive task and similar to SENSS it is a late detection approach. Also the proposal only uses destination address and destination port to find DDoS as compare entropy with a static threshold.

III. Proposed Solution: SAFETY

In this section, we present our proposal called *SAFETY*: an early detection and mitigation technique for TCP SYN flooding attacks in SDN networks. In particular, we start with discussing the foundation elements of SAFETY, i.e., claude shannon's entropy and key observations using data flow entropy (extracting clues from the entropy analysis). Then we present the overall architecture, working methodology, and detailed analysis of *SAFETY*. Finally, we discuss the process of threshold estimation and different possible scenarios for the attack coverage in *SAFETY*.

A. Shannon's Entropy

Shannon's entropy [18] is a concept in information theory which is used to measure the randomness or uncertainty associated with a random variable, in our case <*Destination IP*>. Alternatively, it could also be described as a measure of the information content associated with a random variable. The more random the variable, higher the entropy and vice-versa.

Formal Definition: Equation (1) shows a window, where x_i is the random variable, and y_i represent its frequency. To compute the entropy, Equation (2) is used where $P(x_i)$ denotes the probability of occurrence of each random variable in the set.

$$W = \{(x_1, y_1), (x_2, y_2), (x_3, y_3), \dots, (x_n, y_n)\} \quad (1)$$

$$P(x_i) = y_i/N \quad (2)$$

$$N = y_1 + y_2 + y_3 + \cdots + y_n \quad (3)$$

where, **N** represent the total number of occurrences of all the outcomes. The entropy of a discrete random variable **(X)** that is present in a system is defined as:

$$E(X) = \sum_{i=1}^{n} -P(x_i) log_2 P(x_i) \quad (4)$$

where, **n** represent the number of unique outcomes (i.e., unique $destination_{IP}$). Since entropy falls in the range of [0, $log_2 n$], and it would vary for different window lengths. Therefore, we normalize the entropy to get fall in the same scale of entropy, i.e., in the range of [0, 1], which is independent of the sample size of the window (please refer Equation (5)).

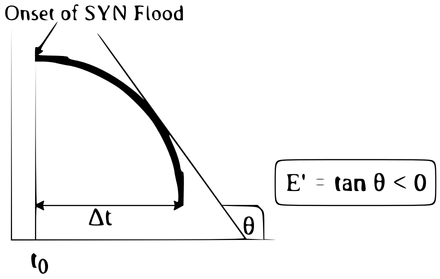


Fig. 3. Nature of slope of entropy w.r.t time.

$$E^N(X) = \frac{E(X)}{log_2 \, n} \quad (5)$$

As mentioned above, entropy is the measure of randomness. SAFETY calculates the entropy of packets in a fixed time window to determine any abnormal behavior. In particular, entropy is calculated on few fields namely destination IP address, destination port, and TCP flags. In case of non-attack scenarios, there are multiple hosts which communicates with each other and their packets identified with a pair consists of destination IP and port, are randomly distributed in the network. However, in presence of adversaries, a particular service irrespective of its location will generate packets with same destination IP and port address. In such scenarios, the randomness is limited due to nature of the attack, hence entropy decreases drastically.

B. Key Observation

Generally, flow entropy will be higher for a normal case of traffic scenario because the number of SYN packets for a destination IP will be of uniform nature. However, once SYN flooding is active, there will be a dominance of attack flows, and consequently, a continuous significance decrease in the entropy would be observed for the duration of the attack period. This is because the concentration of packets for a particular destination IP (victim node) will be very much greater than for the other destination IP. Therefore, it causes a situation of alarm when there is a sudden decrease in entropy [19].

Formal Proof: Assume that SYN flood attack is taking place, and let E(X) denotes entropy at the two consecutive time units of an interval, say Δt (t_0 and $t_0 + \Delta t$) respectively. Now, since we have mentioned that there will be a decrease in the successive entropy as it is also evident from Figure 3. As the difference found in their entropy values is less than a quantity Δ (threshold entropy), hence we can write:

$$E(X)|_{t=t_0} - E(X)|_{t=t_0+\Delta t} \geq \Delta \quad (6)$$

where, *t* represents time, and Δ (Δ >0) represents the threshold entropy at that point of time. Subsequently, we are now in a position to ascertain whether entropy will decrease or increase continuously for the chosen time window. Therefore, we have to test the nature of the derivate of entropy over the

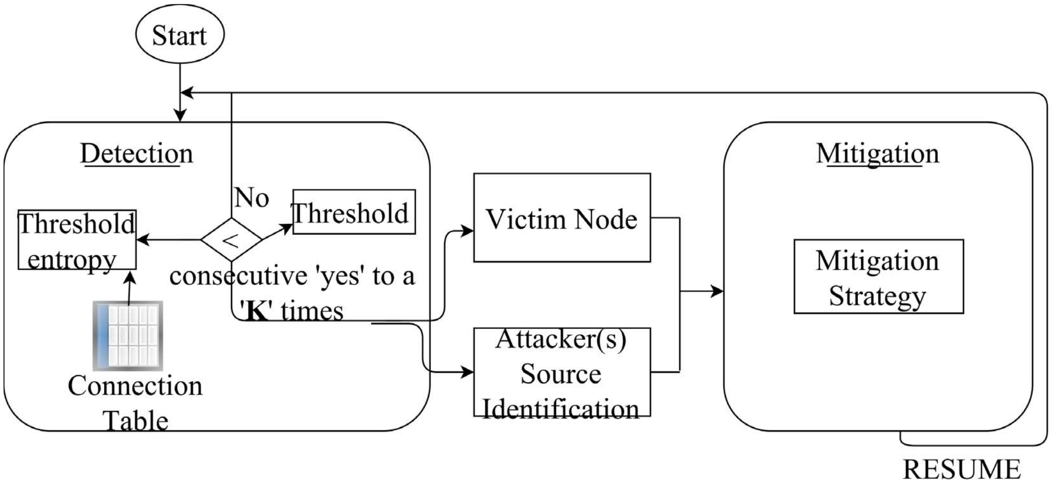


Fig. 4. Overall architecture of *SAFETY*.

assumed time being.

$$E^{'}(X) = \lim_{\Delta t \to 0} \frac{E(X)\big|_{t=t_0+\Delta t} - E(X)\big|_{t=t_0}}{\Delta t} \tag{7}$$

Combining equations (6) and (7) we get,

$$E^{'}(X) \leq -\Delta \tag{8}$$

$$E^{'}(X) < 0 \tag{9}$$

From equations (7) - (9), it could be explained that if persistently there has been a decrement in the value of entropy, and it is found to be lesser than a threshold, it will be confirmed as either malicious traffic or a surge of a flash crowd. Further work has to be done to discriminate the two flows, i.e., benign flows and malicious flows.

C. SAFETY: Architecture and Functionalities

Figure 4 shows the abstract view of the proposed solution: *SAFETY*. The overall architecture of the SAFETY can be viewed as consists of two major components: a *detection unit* and a *mitigation unit*. In detection approach, we compare the entropy of the window with a threshold. A continuous lower value than the threshold could be seen as a suspicious traffic. Detection functionality gives output as a victim node and source of an attacker (DPID of the concerned switch and in_port) as their output. After identifying the attachment point of the malicious host, mitigation strategy would start.

The detailed design of the proposed solution is shown in the flowchart in Figure 5. A connection database (*_DB*(DST_{IP}, *dpid*, in_{PORT}, *Cnt*)) has to be maintained at the controller to keep track of SYN requests directed to a destination node. Since every fresh connection attempt shall start with a segment whose SYN flag is 'set'. *SAFETY* adds this connection primitive to the database with that destination IP field of the concerned packet. If no entry exists for a destination host, an entry would be created else existed host IP's count will be incremented.

Now, suppose that a SYN-ACK packet comes to the controller (as packet_in), corresponding IP's count will be decremented (showing that benign request is attempting). While, the RST generated packets (some host will generate as their IP's has been spoofed by the attacker) signifies malicious traffic is prevalent in the network, so for each RST packet the corresponding IP's count will be further incremented.

Algorithm 1 SAFETY's SYN Flood Attack Detection and Mitigation Procedure

```
Input: PACKET_IN($P_i$), _DB($DST_{IP}$, $dpid$, $in_{PORT}$, Cnt), $\Delta$ t, K, $< E_i >$, $\Delta$
Output: Continuation of attack free environment.
procedure SYN_DDoS_SOLUTION($P_i$, _DB, $\Delta$ t, K, $< E_i >$, $\Delta$)
    if ($P_i.Flags.SYN == 1$) then
        _DB($Dst_{IP}$, dpid, $in_{PORT}$, Cnt) $\leftarrow$ _DB($Dst_{IP}$, dpid, $in_{PORT}$, Cnt+1)
    else
        if ($P_i.Flags.(SYN - ACK) == 1$) then
            _DB($Dst_{IP}$, dpid, $in_{PORT}$, Cnt) $\leftarrow$ _DB($Dst_{IP}$, dpid, $in_{PORT}$, Cnt−1)
        else
            if ($P_i.Flags.RST == 1$) then
                _DB($Dst_{IP}$, dpid, $in_{PORT}$, Cnt) $\leftarrow$ _DB($Dst_{IP}$, dpid, $in_{PORT}$, Cnt+1)
            end if
        end if
    end if
    if (Time Elapsed($\Delta$ t)) then
        Calculate Entropy($E_w$(_DB))
        if ($E_w < \Delta$) then
            Increment AC
            if ( $AC > K$) then
                VICTIM_IDENTITY(_log($Dst_{IP}$))
                LOCATING_ATTACKER(_log($Dst_{IP}$, dpid,$in_{PORT}$))
                MITIGATION(_log(dpid,$in_{PORT}$))
                Reset AC
                Clear _DB
            end if
        else
            if ($< E_i > .isFull()$) then
                THRESHOLD($< E_i >$)
                Reset AC
                Clear _DB
            else
                Insert $E_w$ into $< E_i >$
                Reset AC
                Clear _DB
            end if
        end if
    end if
end procedure
```

After a certain time (Δt), the entropy of window will be calculated and compared with the threshold entropy. If calculated entropy is found to be less than the threshold, then

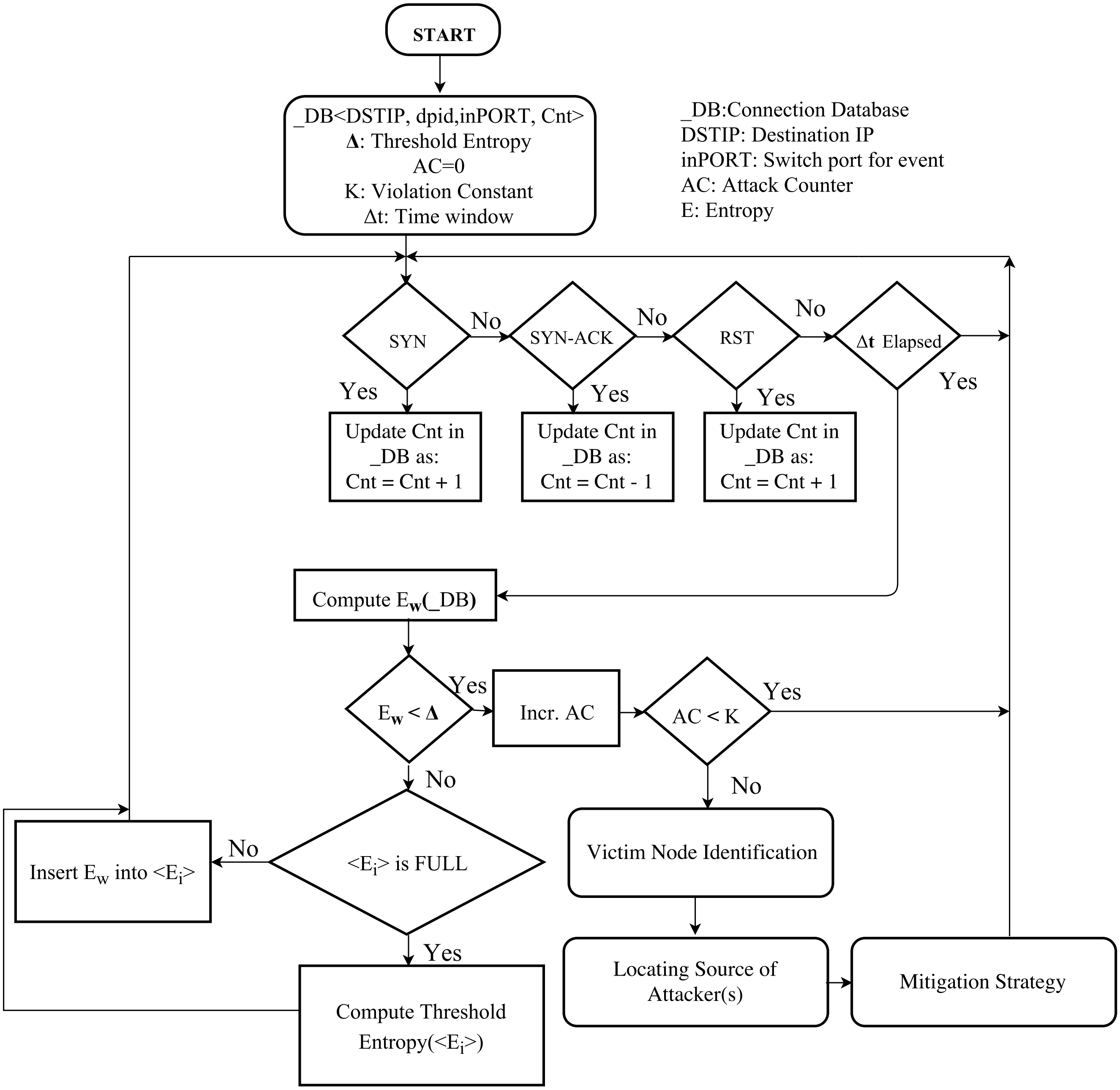


Fig. 5. Flowchart depicting *SAFETY* methodology.

a variable *AttackCounter* (AC) will be incremented. If a violation of window entropy is found to be certain consecutive times (K), then "an attack scenario situation" will be flagged. After the attack condition emergence, then comes the part of victim IP identification followed by locating the source of an attacker and the mitigation process.

In another case, when the entropy of window is found to be greater than the threshold entropy (a case of normal traffic statistics), we will regard that window-entropy as normal entropy and put this entropy to normal values of entropy list $< E_i >$, where i is an instance of normal entropy. If the list gets full, we will recompute entropy threshold based on the newly computed value of mean and standard deviation which we will discuss in detail in the next section.

Time and space complexity: In the absence of any loop, statements between the lines from 1-36, the running time of the proposed Algorithm 1 equals to the running time of its sub-procedures. The victim identification procedure contributes $O(n)$, the attacker identification procedure $O(n)$, and the mitigation strategy contributes $O(1)$. Summing these all, the overall time complexity will be $O(n)$. In addition to some local variables, the algorithm, in turn, depends on the called sub-procedures for estimating the overall space complexity. Hence, summing all these we can observe that the overall space complexity would be $O(n)$.

Victim Identification: Victim identification starts with the process of scanning of log entries for the lookup of the destination IPs whose count value is equal or greater than **K** times.

Algorithm 2 Victim Host Identification Process

```
   Input:_log($Dst_{IP}$), $K$
   Output: ($Dst_{IP}$).
1: procedure VICTIM_IDENTITY(_log($Dst_{IP}$))
2:     for each $Dst_{IP} \in$ _log($Dst_{IP}$) do
3:         if ($Dst_{IP}.count == K$) then
4:             Flag ($Dst_{IP}$) as victim node.
5:         end if
6:     end for
7: end procedure
```

Algorithm 3 Attacker(s) Identification Process

```
   Input: _log($Dst_{IP}$, dpid,$in_{PORT}$), $C$
   Output: <dpid, $in_{PORT}$ >.
1: procedure LOCATING_ATTACKER(_log($Dst_{IP}$, dpid,$in_{PORT}$))
2:     for each dpid, $in_{PORT} \in$ _log($Dst_{IP}$, dpid, $in_{PORT}$) do
3:         if (dpid.$in_{PORT}.count == C$) then
4:             Flag (dpid, $in_{PORT}$) as attacker(s).
5:         end if
6:     end for
7: end procedure
```

If such IP entry exists it denotes its violation number with the threshold entropy. The pseudo-code for the identification steps is given in Algorithm 2.

Time and space complexity: The procedure VICTIM_IDENTITY at line 1 involves a for loop traversing the _log(Dst_{IP}) contributing a total running complexity as *O*(*n*). Apart from _log(Dst_{IP}) which requires a variable size space of *n*, rest are fixed variables which require a constant amount of space. Thus summing up, the space complexity is *O*(*n*).

Locating Source of Attacker(s): Source identification starts with locating the *DPID* and *IN_PORT* of the switch through which malicious traffic is generating. Iteratively search the _log(DST_{IP}, *dpid*, in_{PORT}) entries to look for the <*dpid*,in_{PORT}> and if it is found to be greater than value **C** (i.e., chosen), then this is the attachment point (AP) which is responsible for a major contributor to SYN flooding and the same is reported to the mitigation functionality. The pseudo-code for the same is shown in Algorithm 3.

Time and space complexity: In Algorithm 3, the statement at line 2 involves a *for* loop and due to generalizing, it incurs an *O*(*n*) time complexity. The initializing expression of _log(Dst_{IP}, *dpid*, in_{PORT}) requires a *n* times space and some other local variables add some constant amount. Therefore, the overall space complexity is *O*(*n*).

Mitigation Strategy: After detection phase is over, mitigation process starts. The objective of the proposed solution towards mitigating TCP SYN flood is to disallow the malicious intent while preserving the benign set of traffic. This could be achieved through a SDN-way to mitigate them. *SAFETY* will ensure that from the detected Attachment Point (AP) of the attacker, SYN Flood cannot happen but at the same time allow normal TCP successful connection procedure (characteristic of benign traffic). Some literature has proposed solutions based on the idea of blocking the malicious nodes entirely [5], [6], [12], [14], [15], but as we know a particular node can be made infected through a malicious application (all unknown from the side of the host). The pseudo-code for the mitigation strategy is provided in Algorithm 4.

Algorithm 4 Mitigation Strategy

```
    Input: _log(dpid,$in_{PORT}$), $q$
    Output: Continuation of attack free environment.
 1: procedure MITIGATION(_log(dpid,$in_{PORT}$))
 2:     if ($dpid.in_{PORT}.SYN\_COUNT == q$) then
 3:         Stops ongoing TCP connection until Corresponding SYN-ACK
            arrives from the server
 4:     else
 5:         if ($dpid.in_{PORT}.TCP.flags.(SYN - ACK) == 1$) then
 6:             SYN_COUNT--
 7:             Continue normal TCP procedure
 8:         else
 9:             if ($dpid.in_{PORT}.TCP.flags.(SYN) == 1$) then
10:                 SYN_COUNT++
11:                 Continue normal TCP procedure
12:             end if
13:         end if
14:     end if
15: end procedure
```

For achieving the above mentioned idea, we allow some particular number of concurrent half-connections (SYN request), here we use *q*. Further SYN requests will have to be blocked until ACK of one of the SYN-ACK (from the server side) have returned from the client end-point. This approach, on one hand, it will stop initiating SYN flood from that AP while on the another hand it will let go normal TCP transfer.

Time and space complexity: Mitigation algorithm involves only *if−else* instructions in the absence of looping constructs. Therefore, it will have *O*(1) time complexity. The algorithm works with a space required by _log(*dpid*, in_{PORT}) and some other local variables. Since, the space requirement by _log(*dpid*, in_{PORT}) can go as big as *n*, thus contributing total space complexity of *O*(*n*).

D. Threshold Estimation

Threshold estimation process consists of two parts. First is to find an average entropy value, and the second is to perform a normal distribution analysis on the calculated list of entropy values to get an *apt* value of an entropy and consider it as the threshold. For calculating the average entropy value, we launch traffic in a manner as below:

- 5 % of the traffic directed towards a specific host.
- 10 % of the traffic directed towards a specific host.
- 15 % of the traffic directed towards a specific host.
- 20 % of the traffic directed towards a specific host.
- 25 % of the traffic directed towards a specific host.

Figure 6 describes the theory for getting the normalized entropy values with respect to above mentioned approach. All the five scenarios would provide us a set of points of valid normalized entropy which is subject to a varied intensity of traffic distribution.

Collected samples of normal entropy are subjected to normal distribution analysis of statistical approaches. When scatter-plotting (please refer to Figure 7) these values, we see that it follows a Bell-like curve. Therefore according to "standard deviation and coverage theorem of normal distribution

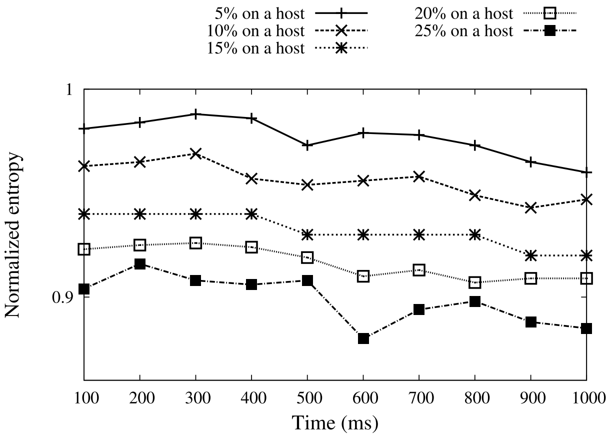


Fig. 6. Distribution of normalized entropy with varying load on a host.

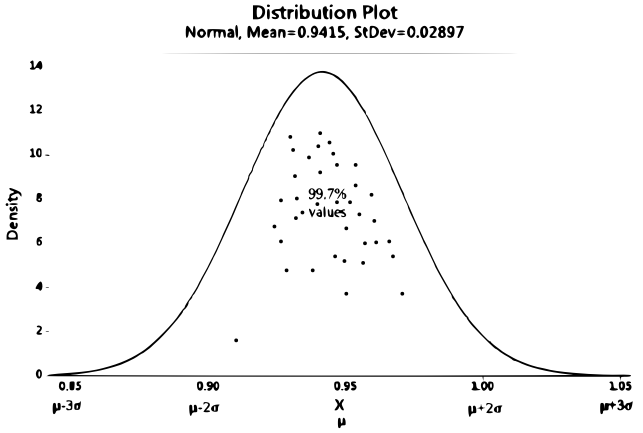


Fig. 7. Density distribution of normal traffic entropy showing Bell distribution.

Algorithm 5 Threshold Estimation

```
Input: $< E_i >$
Output: $\Delta$
procedure THRESHOLD($< E_i >$)
    $\mu \leftarrow 0$
    $\sigma \leftarrow 0$
    $\mu = \sum_{i=1}^{E_i.size()}(E_i)$
    $\sigma = \sqrt{\frac{\sum_{i=1}^{E_i.size()}(E_i-\mu)^2}{E_i.size()}}$
    $\Delta = \mu - 3\sigma$
end procedure
```

theory" about 68% of values come from a normal distribution is found to fall one standard deviation (σ) with the mean (μ), about 95% of the values lie fall two standard deviations, and about 99.7% falls within three standard deviations. This fact is well known as the 68−95−99.7 (empirical) rule, or the *3-sigma rule* [20]. Therefore, mathematically for finding the initial or subsequent points in the threshold will be estimated as described in the Algorithm 5. Explaining algorithm starts with calculating mean and standard deviation from the $< E_i >$ and computing Δ using the below equations.

$$\mu - 3\sigma < E_{th} < \mu + 3\sigma \tag{10}$$

$$\Delta_0 = \mu - 3\sigma \tag{11}$$

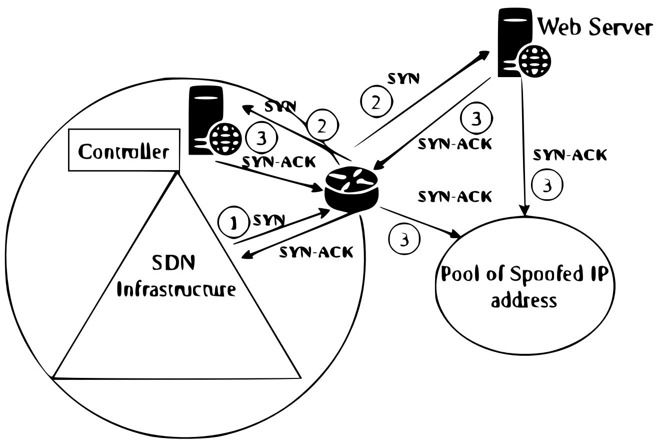


Fig. 8. Working environment of scenario 1.

Time and space complexity Procedure threshold estimation requires a traversal of the entire size of $< E_i >$. Therefore, generalizing it, the algorithm incurs *n* steps for the execution, thus contributing a time complexity of *O*(*n*). Local variables μ and σ require constant space, but $< E_i >$ being a list of size *n* contributes to a total space complexity of *O*(*n*).

Adaptive Threshold: Since network traffic characteristics cannot remain similar all the time. Therefore, with the dynamism of traffic scenario, the point of threshold should change over the time. Hence, our approach towards making them adaptive is described as follow. Whenever any window's entropy is found to be greater than the threshold, we would regard that entropy as one of the normal entropy and put that entropy value into $< E_i >$, as this one indicates one of the benign entropy. Now, when the E_i is full, mean and standard deviation for the set $< E_i >$ (same procedure as described in the threshold point) is calculated. So, a new threshold would be provided to the system to proceed further. By doing this, we have read the characterization of the network traffic situation and act so.

E. Case Study

Scenario 1: The scenario shows the situation when the attacker uses a pool of spoofed IP addresses outside of the SDN-infrastructure environment as shown in Figure 8.

For every SYN request generated from the SDN infrastructure, it would be logged at the controller (count value will get incremented). Now, if there is an SYN request generated from the SDN environment, then corresponding a SYN-ACK reply will come, and it is logged at the Controller and the count value will decremented. However, if SYN request is generated using spoofed IP (count value will incremented), then corresponding SYN-ACK will go over to the spoofed IP thus, the count will go on increasing.

Scenario 2: This scenario shows the situation when the attacker uses a pool of spoofed IP addresses inside of its own SDN-infrastructure environment as shown in Figure 9.

For every SYN request originating from the SDN infrastructure, the count value will get increment (whether malicious or benign request). Now, SYN-ACK from the Web server comes

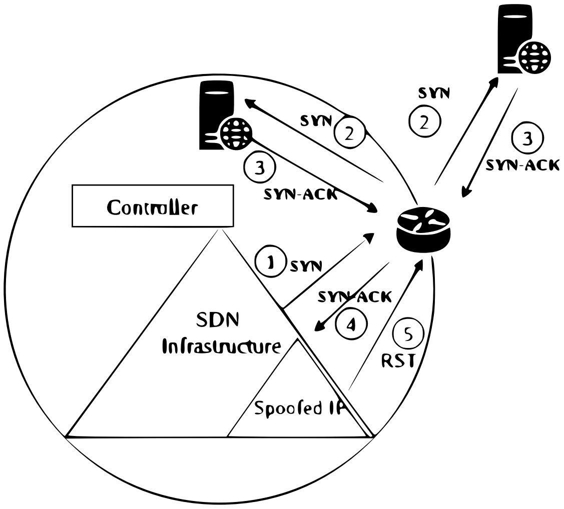


Fig. 9. Working environment of scenario 2.

to the SDN environment, and according to the Algorithm 4 the count will get decrement. However, the crux of the matter is that spoofed machine will now generate RST in response to the unsolicited SYN-ACK coming to them. Therefore, in response to RST packet, the count will get again incremented (true only for a malicious requests). We describe the entire case study and its observations in Table I. The table describes that the solution mechanism can discriminate benign crowd from the flash crowd.

In particular, we are addressing three different nature of traffics and two topological scenarios, hence we had to separate the benign traffic, the flash crowd (which is also benign), and the DDoS traffic (please refer to Table 1). In one topology, spoofed IP addresses do not belong to SDN infrastructure while in next topology it is. In the first scenario for benign traffic when SYN packet arrives the counter gets incremented, and it is decremented when SYN-ACK packet arrives. In case of DDoS, a lot of SYN packets are forwarded and the counter will increase rapidly. No SYN-ACK packet receives at SDN controller because IP addresses that are used for spoofing are outside the infrastructure. Hence any server receiving SYN packet will send SYN-ACK to spoofed IP and these packets will not reach to SDN infrastructure. However, in case of the flash crowd, SYN-ACK is received back, hence the counter decreases accordingly. If, at any period, the counter value is above an adaptive threshold it will trigger the mitigation module. In scenario 2, the only change is that the SYN-ACK sent by the spoofed IPs belonging to the same SDN infrastructure will also come even in case of DDoS. In that case, RST packet is considered to clear the dilemma because any host for whose IP is spoofed to perform the attack will generate RST which confirms that the host has never initiated any communication to the server. This RST packet again increases the counter to make it eligible for DDoS attack.

TABLE I
TRAFFIC DISCRIMINATION IN *SAFETY*

Traffic Nature	Scenario 1	Scenario 2
DDoS	• for SYN, count++;	• for SYN, count++; • for SYN-ACK, count−−; • for RST, count++;
Flash Crowd	• for SYN, count++; • for SYN-ACK, count−−;	• for SYN, count++; • for SYN-ACK, count−−;

TABLE II
ATTACK SETUP PARAMETERS

Resource	**Configuration**
Web Server	Apache 2.4.7
Victim/Benign/Attacker OS	Ubuntu 14.04 LTS 64-bit
Victim/Benign/Attacker Configuration	Intel(R) Core(TM) i5 CPU (4 X 2.40GHz Processor)
Victim Service	HTTP Service
Attack traffic	2000-3000 Packets Per Second
Benign traffic	30-50 Packets Per Second
Network	1 Gbps

IV. Experimental Setup and Performance Analysis

In this section, we have discussed the necessary setup required to carry out the experiments. In particular, Section IV-A discusses the experimental topology. Section IV-B presents different attack manifest and observable parameters to claim for attack scenarios. All the experiments has been done using Mininet *v*2.3.0d1 [21], which emulates a complete network of end-hosts, links, and switches on a single Linux kernel using process-based virtualization. We have run Mininet natively on our Linux machine. We use Open vSwitch (OVS) *v*2.5.0 [22], which is a virtual switch that enables network automation through programmatic extension. Every SDN-switch is identified through a 64-bit DPID (DataPathIdentifier). We configure Floodlight *v*1.2 [23] as the SDN controller. Table II, shows the setup parameters and their values that we have used to create attack scenarios.

A. Experimental Topology

The Web server accepting the TCP connections could be located inside the SDN autonomy or it can be located outside the SDN autonomy (please refer Figure 10). We capture the difference of SYN flooding attack in terms of various networking metrics for both the scenarios.

The entire topology consists of 40 hosts and a Web server. The inter-networking is made up of 11 OpenFlow switches, out of which 4 are configured as an edge switch while the remaining are configured as a core switch. Each edge switch connects 10 end-hosts and each host is connected to the access switch with a dedicated bandwidth of 100 Mbps and switches are interconnected with a bandwidth of 1 Gbps. The Web server is also connected with one of the core switches with a bandwidth of 1 Gbps.

B. Attack Manifest

In this section, we present the attacks manifestation and its quantitative analysis, which are categorized in two

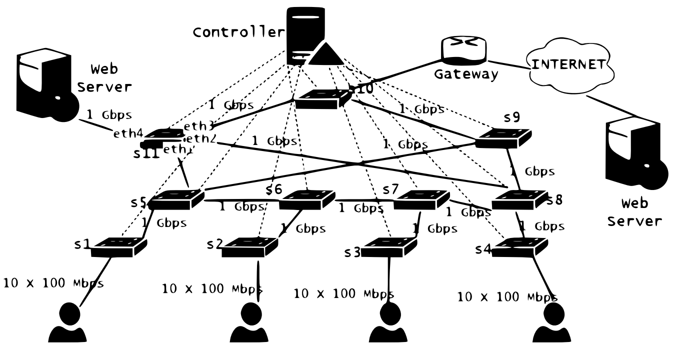


Fig. 10. Experimental topology when Web server is inside (left) and outside (right) of SDN autonomy.

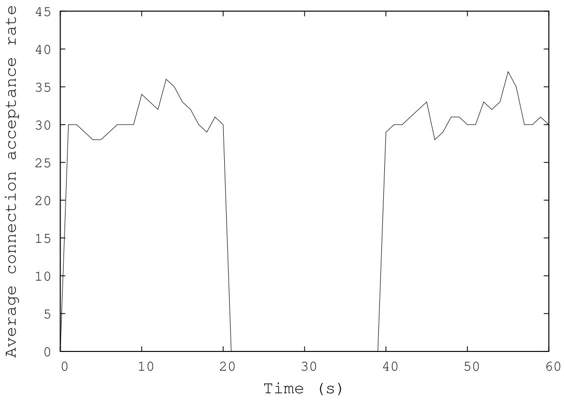


Fig. 11. Average connection acceptance rate over the time.

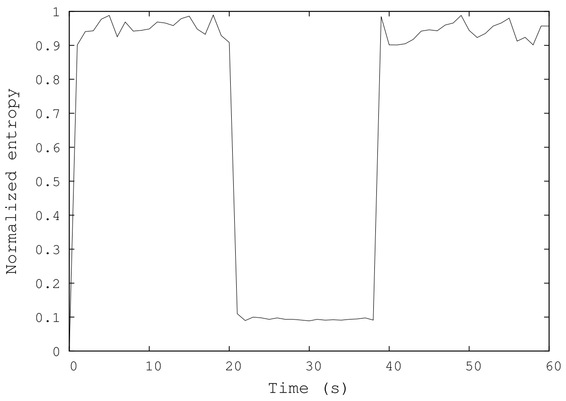


Fig. 12. Observing the time series of average normalized entropy.

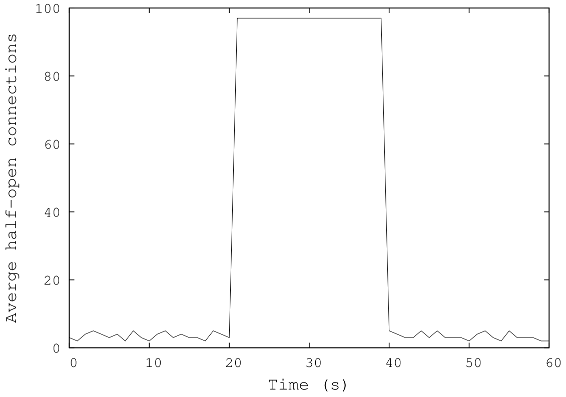


Fig. 13. Average half-open connections at the server end.

dimensions: the traditional network oriented manifests and the SDN-specific manifests. Traditional attack analysis can be thought of as sub-categorized into the client end, the backbone network, and the Web server end. SDN attack analysis includes observation for flow table overflow, and control plane saturation effects. For all these manifest observations, we have allowed normal traffic to be run for starting 20 seconds. After 20 seconds elapsed, attack script is invoked and let execute for 20 seconds. After that, we stop the attack script, and afterwards we let the normal traffic to remain there.

1) Traditional Network-Based Attack Manifestation: Traditional network manifest includes the following evaluation metrics-

1) *Average connection acceptance rate:* It is defined as the average number of connection request attempted to a server and how many it will be acknowledged in packets per second (PPS). Figure 11 shows the connection request acceptance rate over the time. As it is clear that in the absence of attack all connection request of benign hosts is accepted while during the progression of attack connection acceptance for the same is 0.
2) *Average normalized entropy variation:* It is based on destination IP field during a time-window is normalized entropy. Under a normal traffic scenario, owing to a uniform distribution to destination IP, the randomness of a system about the destination IP is quite high. Thus normalized entropy (a quantifier for this state of randomness) is found to be of its maximum value (≈ 1). However, during a case of SYN flood more traffic will be destined to a single node. Thus randomness of the system decreases rapidly. This contributes the factor entropy for destination IP tends to be closer to (≈ 0) as shown in Figure 12.
3) *Half-open connections maintained at the Web server:* Every unacknowledged connection request (SYN packets) is handled by a half-open connection at the Web server. Generally, during a benign traffic course, a connection waits for a very little amount of time in the SYN_RECEIVED state (characteristic of an half-open connection). This is because the response time of a benign request would be very less. So, a client would send an ACK as soon as it receives SYN-ACK. Figure 13 shows the same reasoning as explained above that during legitimate traffic scenario, the number of half-open states is found to be very less (a few, 2 to 5), but the same arose abruptly up (a number of 97 specific to the Apache Web server) during attack duration.
4) *Average CPU utilization at the Web server:* It is defined as the CPU utilization at the Web server due to processing of a lot of SYN requests. Due to an increase in the number of SYN requests at the Web server, processing those requests incurs a high load at their end as quite evident from the Figure 14. During the attack progression, CPU utilization is found to be around 90-95 %.

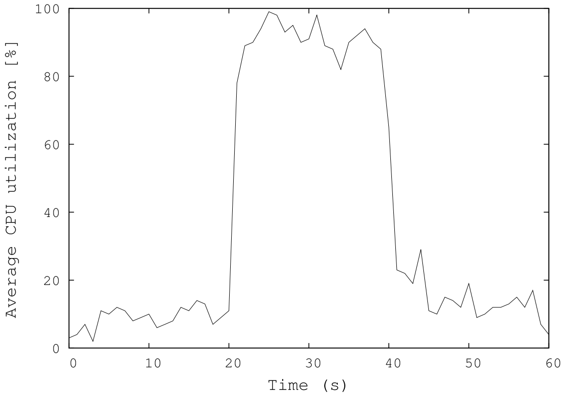


Fig. 14. Average CPU utilization at the Web server.

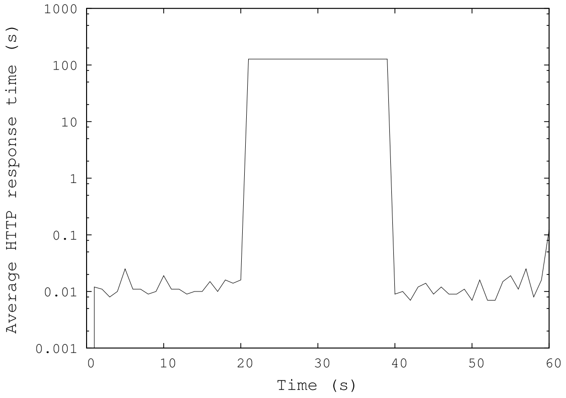


Fig. 16. Measuring average HTTP response time for benign requests.

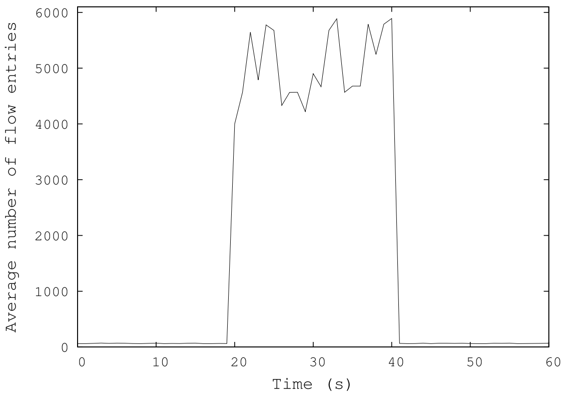


Fig. 15. Average number of flow entries variation with the attack progression.

2) SDN-Based Attack Manifestation: SDN networks manifest includes the following evaluation metrics-

1) *Switch flow table overflow:* Each OpenFlow switch contains one or more flow tables which stores flow entries corresponding to how a packet for a flow will be handled. Generally, TCAM (Ternary Content Addressable Memory) is used to store flow tables. Since memory is finite, therefore every switch is vulnerable to be overflowed if a huge number of flow entries get installed. One of the aims of SYN flooding attacks is to fill a ton of malicious connections at the switches itself. Figure 15 shows the snapshot of the number of flow table entries varying with the attack scenario that leads to an abrupt increment in the flow table entries.
2) *Switch input buffer overflow:* An OpenFlow switch maintains a set of buffers for each Openflow connection. For instance, an Open vSwitch (OVS) keeps a set of 256 buffers for handling Packet_In messages. For each generated Packet_In message, OVS assign a buffer_id. Now, a small part of packet header goes to a controller for processing. As long as the controller sends a Packet_Out or a Flow_Mod message or 1 second elapsed (whichever earlier), it will hold the buffer for that Packet_In. Thus, packet buffering allows a small part of the packet header to be moved to the controller instead of the whole header. However, since due to its finiteness, the buffer pool can be overflowed due to SYN flood attacks.
3) *Control plane saturation:* The extensive communication between the controller and the OpenFlow switches creates a bottleneck on the link between them. Once the link got saturated, the switches cannot get benign flows installed. Therefore, we could observe a significant delay in the HTTP response time for benign client requests, once SYN flooding is in action. Figure 16 shows the response time measured for the benign hosts over the time. A Web page fetching response rate is in the order of milliseconds, but attack progression has made to raise the response time waiting indefinitely until a benign host would give up.

C. SAFETY Performance Evaluation

In this section, we analyze the results of our proposed solution to evaluate its performance against the adverse effects (as shown in Section IV-B) caused by TCP SYN flooding attack on SDN networks. Evaluation of our solution encompasses the assessment of the following three parameters:

- *Average attack detection time:* It is the time difference between the instant an attack is detected and the initiation of the first SYN packet by the attacker.
- *Average delay to establish a genuine request in the presence of attack:* It is the time taken by a benign host to get the HTTP response back from the server.
- *Average CPU consumption by the controller:* It is the CPU utilization of the Process ID (PID) associated with the instance of the running controller. In our case, we monitored the CPU utilization attached with the Floodlight controller.

For conducting the experiments, we run a number experiments with different values of Violation Constant (K) and Time Window (i.e., Δt, represented as t in figures 17 and 20) in the following two scenarios: (i) varying the number of SYN requests, and (ii) varying the number of attackers.

1) Scenario 1: For this experiment scenario, the number of attackers is set to 28, and each attacker sends SYN packets varying from 60 to 260.

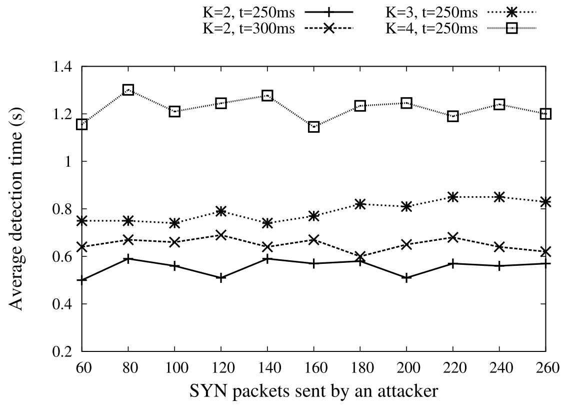


Fig. 17. Average attack detection time with varying malicious SYN packets.

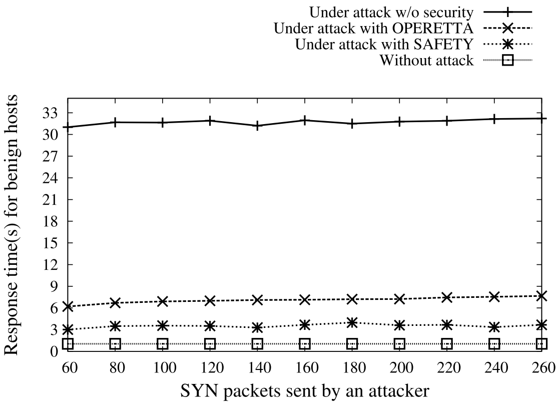


Fig. 18. Average delay to establish an HTTP connection with varying malicious SYN packets.

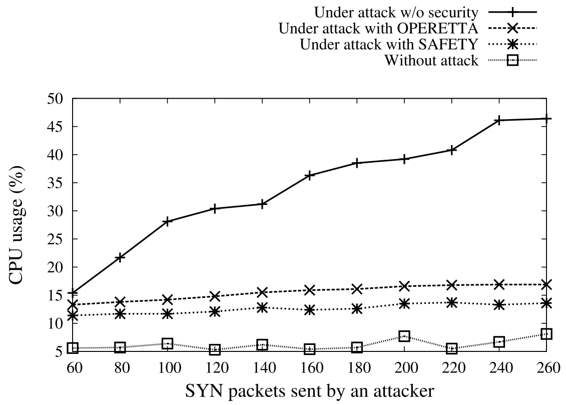


Fig. 19. Average CPU consumption with varying malicious SYN packets.

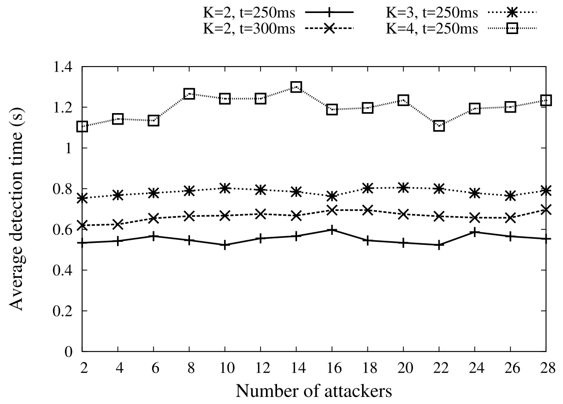


Fig. 20. Average attack detection time with varying attackers.

- Attackers (employing IP spoofing accordingly) ⇒ $s1<h1-h10>$, $s2<h11-h20>$, $s3<h21-h28>$, here $s1<h1-h10>$ means that hosts $<h1, h2, h3, \ldots, h10>$ are connected with switch $s1$,
- Legitimate hosts $h29$, $h30$ are connected with switch $s3$ ⇒ $s3<h29, h30>$
- IP address of host $h31$ to $h40$ is used as pool of spoofed IP ⇒ $s4<h31-h40>$
- Web server host $h41$ is connected with switch $s11$ ⇒ $s11<h41>$
- *Average attack detection time:* The detection time depends on the concurrent number of SYN requests directed towards a victim end. This, in turn, will be feed through the product of K and Δt. Hence, whenever, the product will be increased, we will see an increase in the detection time as it can be shown in Figure 17. One important observation that we infer from the Figure 17 is that increasing the K affects a significant contribution in the increment of detection time as this implies one more iteration to be done to claim for an attack.
- *Average delay to establish an HTTP request:* Once an attacker is identified, the SAFETY will apply the mitigation procedure for the list of attackers retrieved from the attacker identification procedure. The above fact follows Figure 18 observation that fixing the number of attackers (i.e., 28), we would not observe any noticeable change in the delay for a successful connection setup. As it can been seen from Figure 18 that response time for SAETY increases from 0.5 to 3 seconds, but it is still half than OPERETTA (i.e., 6 seconds). This is because of the use of entropy-based adaptive thresholds that are being used in SAFETY to detect an attacker and its lower time and space complexities (please refer to Section III-C).
- *Average CPU utilization by the controller:* Increasing the request rate will certainly tend to incur a noticeable overhead at the controller as shown by the red curve in Figure 19. The increment in CPU utilization is due to the message exchange between the controller and the switches. For a slower rate of connection requests, generally there would be a small gap in the CPU usage between the attack scenario and SAFETY scenario.

2) Scenario 2: For the second experiment scenario, a variable number of attackers (ranging from 2 to 28) each sending a fixed number of SYN packets (i.e., 100).

- Variable number of Attackers (employing IP spoofing accordingly) from the set $<h1, \ldots, h28>$
- Legitimate hosts ⇒ s3(h29, h30)
- Pool of spoofed IP ⇒ s4(h31-h40)
- Web server host ⇒ s11(h41)
- *Average attack detection time:* As mentioned in the previous section that the average detection time depends

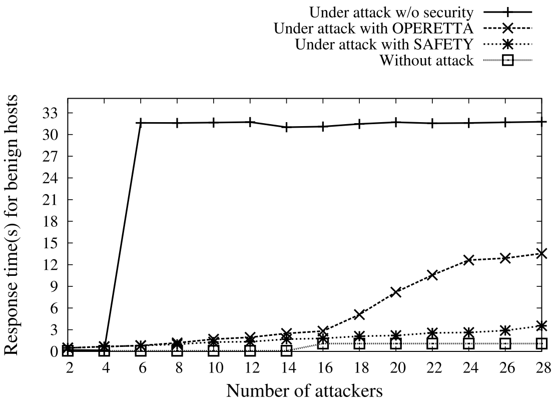


Fig. 21. Average delay to establish an HTTP connection with varying attackers.

on the product of K and Δt. Hence, as expected whenever the product constant increases, the average detection time also increases as shown in Figure 20. Furthermore, varying the number of attackers will not change the impact on detection time as long as attackers sufficiently inject a sufficient time of fake SYN packets constantly.

- *Average delay to establish an HTTP request:* Increasing the number of attackers will tend to increase the response time for a benign client to retrieve a response from the Web server as shown in Figure 21. The reason for this behavior is that each attacker need to deal with the SAFETY, only after it get cleared, a successful connection would get established.
- *Average CPU utilization by the controller:* Increasing the number of attackers with fixed sending rate will in turn increase the number of Packet_In message that need to be exchanged between the controller and the OpenFlow switches. Therefore, the controller with increased request rate will have to handle more and more such messages which increases the CPU overhead. We infer that for less number of attackers the overhead at the controller will be more compared to the standard (without defense) mechanism as SAFETY would have to do some additional work. Further, the increase in the number of attackers, in turn increases the number of connections which will outperform the standard approach as it is shown in Figure 22.

We use the Receiver Operating Characteristic (ROC) curves to show the trade-off between the true positive rate (TPR) and false positive rate (FPR) by varying the threshold entropy (Δ). The FPR (i.e., 1-specificity) is the percentage of malicious SYN packets which are incorrectly identified as benign SYN packets, and the TPR (i.e., sensitivity) is the percentage of benign packets which are correctly identified as benign packets. These metrics are measured under DDoS TCP SYN flooding attacks to one victim host. The traffic parameters are as follows (i) Scenario 1: average traffic rate 50 Mbps from which 200 to 300 packets per second is attacker's traffic, and (ii) Scenario 2: average traffic rate 100 Mbps from which 1000 to 1500 packets per second is attacker's traffic. The monitoring interval Δt is 3 sec, violation constant K is set to 3 and attack counter AC is 2. As shown in Figure 23, we can see that SAFETY can achieve 100% TPR while has approximately 27% FPR for both the target scenarios. SAFETY works better in high traffic scenario, i.e., 100 Mbps. In this scenario, the regular traffic is very low rate and it is easy to distinguish the attack. In particular, due to the presence of higher malicious traffic (i.e., 1000 to 1500 packets per second), there will be a persistent decrement in the value of the calculated entropy, and it is found to be lesser than a threshold. Hence, it is easy to distinguish the attack in high traffic scenario because it is beneficial to randomize the traffic and support our entropy calculations.

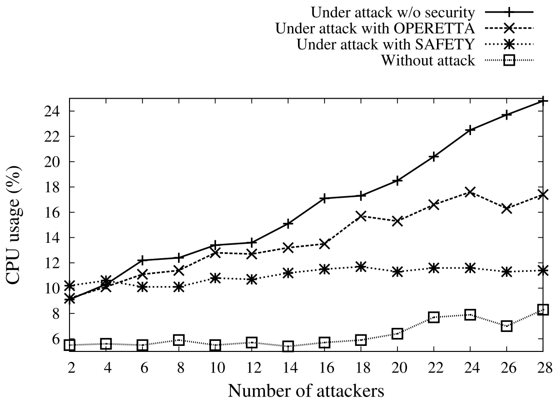


Fig. 22. Average CPU consumption with varying attackers.

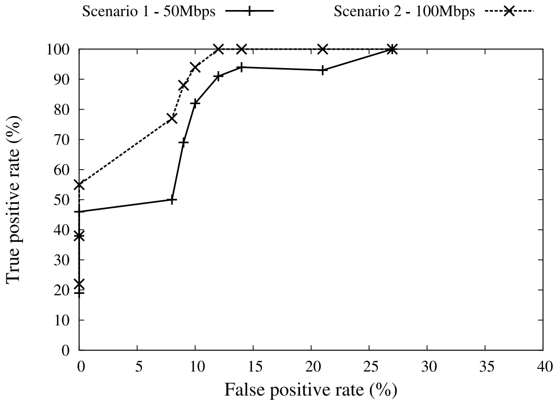


Fig. 23. ROC curves for the 50 Mbps and 100 Mbps cases scenario.

TABLE III
DELAY (IN SECONDS) FOR A LEGITIMATE HOSTS CONNECTION UNDER NO ATTACKS

W/o security	OPERETTA	SAFETY
0.0345	1.10	0.1455

Finally, Table III shows the performance of SAFETY in a normal scenario i.e., in the absence of an attacker. Table III depicts that SAFETY exhibits lower delay in connection set up when compared with OPERETTA. This is due to its low time and space complexity.

V. CONCLUSION

This paper presents an entropy-based early detection and mitigation technique against TCP SYN flooding attacks in

SDN networks. We show that based on traffic features, such as clubbing of destination IP addresses along with TCP flags and following a sequence of time-based windowing of packets, one can efficiently calculate the entropy to measure the degree of randomness of the packets that are received at the SDN controller. By computing the entropy and further comparing it with the threshold at that instance (adaptive), we shows through emulation results, that *SAFETY* is capable of early detection as well as mitigation of the attack at the edge switches itself.

Since the entropy-based solution has been found to have inherent limitation for the distribution or skewness to a certain random variable. Hence, in this paper, we have to make the assumption with a single victim destination node, but in future, we would like to work for multiple destination victims. Since every sort of TCP connection is directed towards the controller (for making a centralized statistics) further work will be to lessen the number of messages exchanged and also the controller from being saturated from these connection requests. Additionally, we plan to use the realistic traffic for security experimentations because it more randomized than the synthetic traffic (e.g., wget).

References

[1] D. Tuncer, M. Charalambides, S. Clayman, and G. Pavlou, "Adaptive resource management and control in software defined networks," *IEEE Trans. Netw. Service Manag.*, vol. 12, no. 1, pp. 18–33, Mar. 2015.

[2] N. McKeown *et al.*, "OpenFlow: Enabling innovation in campus networks," *SIGCOMM Comput. Commun. Rev.*, vol. 38, no. 2, pp. 69–74, Mar. 2008.

[3] J. Lemon, "Resisting SYN flood DoS attacks with a SYN cache," in *Proc. BSD BSDC*, San Francisco, CA, USA, 2002, p. 10. [Online]. Available: http://dl.acm.org/citation.cfm?id=1250894.1250904

[4] "TCP SYN flooding attacks and common mitigations," Internet Eng. Task Force, Fremont, CA, USA, RFC 4987, 2007. [Online]. Available: https://tools.ietf.org/html/rfc4987/

[5] S. M. Mousavi and M. St-Hilaire, "Early detection of DDoS attacks against SDN controllers," in *Proc. IEEE Int. Conf. Comput. Netw. Commun. (ICNC)*, Garden Grove, CA, USA, 2015, pp. 77–81.

[6] R. Wang, Z. Jia, and L. Ju, "An entropy-based distributed DDoS detection mechanism in software-defined networking," in *Proc. IEEE TrustCom/BigDataSE/ISPA*, 2015, pp. 310–317. [Online]. Available: http://dblp.uni-trier.de/db/conf/trustcom/trustcom2015-1.html#WangJJ15

[7] J. Postel, "Transmission control protocol," Internet Eng. Task Force, Fremont, CA, USA, RFC 793, Sep. 1981. [Online]. Available: http://www.rfc-editor.org/rfc/rfc793.txt

[8] R. Braga, E. Mota, and A. Passito, "Lightweight DDoS flooding attack detection using NOX/OpenFlow," in *Proc. IEEE Local Comput. Netw. Conf.*, Oct. 2010, pp. 408–415.

[9] T. Chin, X. Mountrouidou, X. Li, and K. Xiong, "Selective packet inspection to detect DoS flooding using software defined networking (SDN)," in *Proc. IEEE 35th Int. Conf. Distrib. Comput. Syst. Workshops*, Jun. 2015, pp. 95–99.

[10] S. Shin, V. Yegneswaran, P. Porras, and G. Gu, "AVANT-GUARD: Scalable and vigilant switch flow management in software-defined networks," in *Proc. ACM SIGSAC Conf. Comput. Commun. Security (CCS)*, 2013, pp. 413–424.

[11] M. Ambrosin, M. Conti, F. D. Gaspari, and R. Poovendran, "LineSwitch: Tackling control plane saturation attacks in software-defined networking," *IEEE/ACM Trans. Netw.*, vol. 25, no. 2, pp. 1206–1219, Apr. 2017.

[12] M. Nugraha, I. Paramita, A. Musa, D. Choi, and B. Cho, "Utilizing OpenFlow and sFlow to detect and mitigate SYN flooding attack," *J. Korea Multimedia Soc.*, vol. 17, no. 8, pp. 988–994, 2014.

[13] M. Dhawan, R. Poddar, K. Mahajan, and V. Mann, "SPHINX: Detecting security attacks in software-defined networks," in *Proc. NDSS*, 2015, pp. 1–15.

[14] S. Fichera, L. Galluccio, S. C. Grancagnolo, G. Morabito, and S. Palazzo, "OPERETTA: An OpenFlow-based remedy to mitigate TCP SYNFLOOD attacks against Web servers," *Comput. Netw.*, vol. 92, pp. 89–100, Dec. 2015. [Online]. Available: http://dblp.uni-trier.de/db/journals/cn/cn92.html#FicheraGGMP15

[15] R. Mohammadi, R. Javidan, and M. Conti, "SLICOTS: An SDN-based lightweight countermeasure for TCP SYN flooding attacks," *IEEE Trans. Netw. Service Manag.*, vol. 14, no. 2, pp. 487–497, Jun. 2017.

[16] M. Yu, Y. Zhang, J. Mirkovic, and A. Alwabel, "SENSS: Software defined security service," Presented at the Open Netw. Summit (ONS), Santa Clara, CA, USA, 2014. [Online]. Available: https://www.usenix.org/conference/ons2014/technical-sessions/presentation/yu

[17] K. Giotis, C. Argyropoulos, G. Androulidakis, D. Kalogeras, and V. Maglaris, "Combining OpenFlow and sFlow for an effective and scalable anomaly detection and mitigation mechanism on SDN environments," *Comput. Netw.*, vol. 62, pp. 122–136, Apr. 2014.

[18] C. E. Shannon, "Prediction and entropy of printed English," *Bell Syst. Tech. J.*, vol. 30, no. 1, pp. 50–64, Jan. 1951.

[19] Y. Tao and S. Yu, "DDoS attack detection at local area networks using information theoretical metrics," in *Proc. 12th IEEE Int. Conf. Trust Security Privacy Comput. Commun.*, Jul. 2013, pp. 233–240.

[20] *Wikipedia*. [Online]. Available: https://en.wikipedia.org/wiki/68%E2%80%9395%E2%80%9399.7_rule/

[21] M-Team. (2017). *Mininet*. [Online]. Available: http://mininet.org/

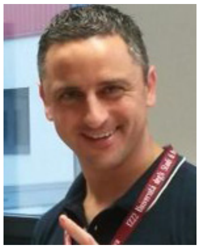
[22] LFC Project. (2017). *Open vSwitch*. [Online]. Available: http://openvswitch.org/

[23] BS Networks. (2017). *Project Floodlight*. [Online]. Available: http://www.projectfloodlight.org/

Prashant Kumar received the master's degree in computer science and engineering from the Malaviya National Institute of Technology Jaipur, India, in 2017. He is a Software Engineer with Milvik (BIMA), Bengaluru, India.

Meenakshi Tripathi (M'14) received the Ph.D. degree from the Malaviya National Institute of Technology, Jaipur, India, in 2015, where she is an Associate Professor. She has published over 30 research papers in refereed international and national journals and proceedings of national and international conferences. She has presented her research ideas by participating in various conferences within India and outside India, like China, Canada, USA. She is researching on various research projects funded by the Department of Science and Technology, Ministry of Electronics and Information Technology, India, of worth around Rs. 50 Lakhs. Her research areas are wireless sensor networks, software defined networks, and Internet of Things. She is a reviewer for several renowned journals, such as the *International Journal of Communication System* (Wiley), *Security and Communication Networks* (Hindawi), and *Wireless Communications*. She has also been part of several international conferences, such as SPACE 2015, ICISS 2016, and ICSP 2019. She is an upright member of Association of Computing Machinery. She is also the Chairman of Computer Society of India (Jaipur Chapter).

Ajay Nehra received the Bachelor of Engineering degree in computer engineering from the University of Rajasthan in 2008 and the M.Tech. degree in computer science and engineering from the Central University of Rajasthan, India, in 2012. He is currently pursuing the Doctoral degree with the Department of Computer Science and Engineering, Malaviya National Institute of Technology, Jaipur, India. He has been awarded scholarships for both the master's and Doctoral degrees from the Ministry of Human Resource Development, Government of India. His current research area includes software defined networks, information security, and network security.



Mauro Conti is Full Professor at the University of Padua, Italy, and Affiliate Professor at the University of Washington, Seattle, USA. He obtained his Ph.D. from Sapienza University of Rome, Italy, in 2009. After his Ph.D., he was a Post-Doc Researcher at Vrije Universiteit Amsterdam, The Netherlands. In 2011 he joined as Assistant Professor the University of Padua, where he became Associate Professor in 2015, and Full Professor in 2018. He has been Visiting Researcher at GMU (2008, 2016), UCLA (2010), UCI (2012, 2013, 2014, 2017), TU Darmstadt (2013), UF (2015), and FIU (2015, 2016). He has been awarded with a Marie Curie Fellowship (2012) by the European Commission, and with a Fellowship by the German DAAD (2013). His research is also funded by companies, including Cisco and Intel. His main research interest is in the area of security and privacy. In this area, he published more than 200 papers in topmost international peer-reviewed journals and conference. He is Area Editor-in-Chief for IEEE COMMUNICATIONS SURVEYS & TUTORIALS, and Associate Editor for several journals, including IEEE COMMUNICATIONS SURVEYS & TUTORIALS, IEEE TRANSACTIONS ON INFORMATION FORENSICS AND SECURITY, and IEEE TRANSACTIONS ON NETWORK AND SERVICE MANAGEMENT. He was Program Chair for TRUST 2015, ICISS 2016, WiSec 2017, and General Chair for SecureComm 2012 and ACM SACMAT 2013. He is Senior Member of the IEEE.

Chhagan Lal received the bachelor's degree in computer science and engineering from MBM Engineering College, Jodhpur, India, in 2006, the master's degree in information technology with specialization in wireless communication from the Indian Institute of Information Technology, Allahabad, in 2009, and the Ph.D. degree in computer science and engineering from the Malaviya National Institute of Technology, Jaipur, India, in 2014. He is a Post-Doctoral Fellow with the Department of Mathematics, University of Padua, Italy. He has been awarded the Canadian Commonwealth Scholarship in 2012 under Canadian Commonwealth Scholarship Program to work in the University of Saskatchewan, Saskatoon, SK, Canada. His current research areas include blockchain analysis, security in wireless networks, software-defined networking, underwater acoustic networks, and context-based security solutions for Internet of Things networks.