EAI Endorsed Transactions
on Security and Safety

Research Article EAI.EU

Kernel-Space Intrusion Detection Using Software-Defined Networking

Tommy Chin[1], Kaiqi Xiong[2,*], and Mohamed Rahouti[2]

[1]Rochester Institute of Technology, Rochester, New York, 14623, USA
[2]University of South Florida, Tampa, Florida 33620, USA

Abstract

Software-Defined Networking (SDN) has encountered serious Denial of Service (DoS) attacks. However, existing approaches cannot sufficiently address the serious attacks in the real world because they often present significant overhead and they require long detection and mitigation time. In this paper, we propose a lightweight kernel-level intrusion detection and prevention framework called KernelDetect, which leverages modular string searching and filtering mechanisms with SDN techniques. In KernelDetect, we sufficiently utilize the strengths of the Aho-Corasick and Bloom filter to design KernelDetect by using SDN. We further experimentally compare it with SNORT and BROS, two conventional and popular Intrusion Detection and Prevention System (IDPS) on the Global Environment for Networking Innovations (GENI), a real-world testbed. Our comprehensive studies through experimental data and analysis show that KernelDetect is more efficient and effective than SNORT and BROS.

Received on 01 May 2018; accepted on 02 June 2018; published on 09 October 2018

Keywords: Intrusion Detection and Prevention Systems (IDPS), Software-Defined Networking (SDN), Bloom Filter, Aho-Corasick, Security

doi:10.4108/eai.13-7-2018.155168

1. Introduction

Software-Defined Networking (SDN) has become an emerging technology in computer science and engineering. It has been applied to solve a variety of real-world problems such as cloud resource managements, network provisioning, storage management, network intelligence and monitoring, distributed service control and cloud integration, high performance computing, and an Intrusion Detection and Prevention System (IDPS). For example, a network administrator may leverage multiple options in optimizing and securing a computer network by implementing a De-Militarized Zone (DMZ), integrating a Quality-of-Service (QoS) rule set, or installing an in-line IDPS. SDN can effectively help solve these real-world problems due to the two key features of SDN including its programmability and global-view of an environment. Researchers have leveraged these two features to minimize end-to-end delay [5], control network traffic [17], and security attacks [8].

However, it has been proved that SDN itself is vulnerable to various adverse attacks. Hong, et al. [23] identified threats including Denial of Service (DoS) in SDN and examined DoS attacks under the environment of eight different SDN controllers, but there still remain grand challenges to detect and mitigate them. Traditional approaches to conquering DoS attacks is through the use of Intrusion Detection Systems (IDS) and Intrusion Prevention Systems (IPS). However, existing IDPS solutions present serious concerns including system performance [14], network communication constraints [15], and detection validity [35]. Additionally, IDS detection methods present a critical flaw to identify new or unknown network attacks due to limiting threat signatures and comparison approaches. Recent studies have suggested a variety of threat mitigation and detection solutions including FloodGuard [14], SPHINX [15], and an entropy-based solution [35], but none of them, to the best of our knowledge, has studied a modular kernel-level IDPS approach within SDN environments. Moreover, existing IPDPS solutions such as SNORT [19]

*Corresponding author. Email: xiongk@usf.edu

and BRO [20] suffer from threat detection and mitigation time so as to cause significant performance overhead.

In this paper, we consider an environment like Science DMZ where there is a need to high-speed network access to computation and storage for science research. We propose a lightweight modular-based filtering approach inspired by Amann et al. [16] and Mekky et al. [17], called *KernelDetect*, to detect and mitigate security attacks in an SDN environment. Specifically, KernelDetect is an independent application-plane network Test Access Point (TAP) approach using Switch Port Analyzer (SPAN) interfaces [12] on SDN switching devices. Moreover, in KernelDetect, we consider the fact that the Aho-Corasick and Bloom filter are exact string matching and partial matching techniques, respectively. We sufficiently utilize the strengths of these techniques to design KernelDetect by using SDN. By using a modular approach as a key component, KernelDetect can interchange the technique for string matching in addition to updating its signatures while providing threat mitigation capabilities within a kernel space. As we know, IDS signature methods are to compare a list of given strings or a set of rules with incoming network traffic signatures. In this paper, the proposed KernelDetect provides the ability to dynamically update the rule set in SDN environments in which we can optimize traffic inspection when detecting network threats.

To understand the efficiency and effectiveness of KernelDetect, we experimentally compare it with SNORT [19] and BRO [20], two conventional and popular IPDPS on a real-world testbed - Global Environment for Network Innovations (GENI) [18] to conduct our real-world experimental evaluation. In this comparison, KernelDetect leverages the Aho-Corasick [39] algorithm and Bloom filter [40] with SDN. To provide hybrid network communications, we utilize D-ITG [21] and iPerf [22] as traffic generation software for normal user data in the SDN experiments. To mix normal user traffic with malicious ones, we implement DoS attacks [35] in our threat detection and mitigation experiments. We further implement KernelDetect in an environment driven by Floodlight [24] using Representational State Transfer (REST) Application Program Interface (API) as our method of communication for KernelDetect to mitigate adverse threats and attacks. To be precise, we focus on DoS attacks in this research. That is, we comparatively examine KernelDetect over traditional IDPS technologies, SNORT and BRO, for the detection and mitigation of DoS attacks in a real-world testbed environment where we test various numbers of packets ranging from 100K to 500K and examine SYN flooding attack with different packet sizes and sampling times. In our extensive experiments, we measure the average load of system resources, inspection time, mitigation time, true positive, false positive, and false negative. Our comprehensive studies through experimental results and analysis show that KernelDetect is more efficient and effective than SNORT and BROS. That is, KernelDetect has great performance in intrusion detection and prevention.

In this research, we have made the following main contributions.

1. DoS has been identified as a serious attack in an SDN environment [23]. We present KernelDetect, a lightweight kernel-level IDPS approach to thwarting DoS threats with the ability to interchange string matching detection mechanisms between the Aho-Corasick algorithm [39] and the Bloom filter algorithm.

2. Existing IDPS tools such as SNORT and BRO utilize a culmination of user and kernel space due to the necessary user interaction needed to configure both solutions. Contrary to existing conventional studies, KernelDetect is a pure kernel-space solution. Furthermore, the default installations of SNORT and BRO provide many detection rules for their respective systems. The more number of rules we use, the more performance overhead is added. KernelDetect has a much less overhead compared to SNORT and BRO.

3. We leverage the common architecture of Science DMZ with SDN technologies to develop KernelDetect. Thus, KernelDetect applies to Science DMZ, and it can enhance data-driven research in academia and national laboratories, and other related applications in industry and government agencies.

4. As SNORT [19] and BRO [20] are traditional IDS solutions, we experimentally evaluate KernelDetect against the two well-known kernel-space and user-space detection tools in a real-world testbed, whereas many existing studies are evaluated either through a simulator such as Mininet [13] or in a lab environment whose results are often away from realistic.

The rest of this paper is organized as follows. Section 2 gives our research background with the discussion of our challenges in the research. Section 3 presents related work and Section 4 describes threat models with an attack vector, Section 5 presents the architectural design to address our research challenges. In Section 6, we state the methodology of our experimental evaluation with the experimental setup of KernelDetect and experimental results. To the end,

Section 7 summarizes our research results and provides future work.

2. Research Background and Challenges

This research aims to design and develop an efficient IDS solution on a kernel space. In this section, we present a brief background of kernel-space detection techniques and discuss the main challenges of our research studied in this paper.

2.1. Kernel-Space Detection Background

Although system applications and services and a traditional IDS exploits both user and kernel-spaces of computing [15], existing kernel-space solutions present limited visibility. The majority of them are based on a culmination of kernel and user spaces to characterize their network threats and anomalies. Furthermore, in an SDN environment, SDN switches (e.g., Open vSwitch (OVS)) are linked to both user and kernel-spaces and they inquire network data packets from raw sockets on their corresponding operating system to carry the proper packet data to the SDN switching service. Using a kernel space guarantees full knowledge of threads, high-performance, and low overhead. However, it preserves an instability concern due to kernel panic, and therefore introduces the following challenges.

2.2. Research Challenges and Assumption

Common approaches to detect and mitigate adverse threats is through the use of an IDPS. One major issue of such a technique is through user-space utilization. Moreover, numerous IDS solutions rely on user-space interfaces to allow administrators to manage and maintain the various services that are implemented to identify and thwart malicious attacks.

1. *Kernel Panic:* The first challenge through the use of a kernel space is when a system is panic. Commonly, when a kernel module or a kernel-space application generates an erroneous issue such as a programming bug or a buffer overflow, a panic occurs such that the operating system is no longer function to provide service to the end user. When such an event occurs, a sequence of recovery mechanisms is executed such as memory dumping and a total system restart. We identify this challenge as a significant area to address as KernelDetect resides purely on a kernel-space. We identify this challenge as a significant area to address as the operation of KernelDetect resides purely in kernel-space and that if KernelDetect malfunctions or generates a programmatic error, a kernel panic would occur.
2. *Root Access and System Vulnerability:* Using kernel-space detection requires a significant level of system access to identify such malicious traffic. This level of access is known as root-access and proposes a serious challenge if the IDPS solution [11] were to be compromised or exploited.

Moreover, to both inspect traffic and determine adverse behaviors, elevated access is required on such service to gain a control of raw sockets on an operating system. Using traditional IDS solutions such as BRO and SNORT, service accounts are created to secure the system from exploitation through techniques such as chroot and jailing. These concerns present the second challenge.

The first challenge is that IDPS solutions [37], unarguably, need to provide robustness against buffer overflow, resiliency to exploitation schemes and attacks [27], and ability to manage and inspect large quantities of network traffic in an efficient manner, which depends on the use of a operating system's raw socket feature. Moreover, under a scenario of a DoS attack, excessive packet drops would occur as the system is incapable of dealing with the quantity of network packets properly [1], and therefore the overhead and congestion caused by the DoS attack results in a significant delay as the inspection system places network packets into a queue waiting for their evaluation. This temporary queue would significantly increase overtime, which might result in a buffer overflow. The simplest resolution to alleviate the overflow would be firmly dropping packets in order to limit resource exhaustion of the inspection system. However, this procedure would be problematic if the network traffic has a certain level of urgency or priority (i.e., QoS) [5, 9, 41, 45]. Another serious concern regarding the utilization of kernel space detection is that the application configuration requires root-level privileges on the IDPS solution and therefore, introduces security concerns if the system becomes compromised.

In the second challenge, kernel-space IDPS solutions demand an elevated user to have the root-access to acquire accessibility to a variety of raw socket communication in order to deeply inspect and evaluate incoming packets [38]. Although such access privileges are indispensable for traffic inspection, they presents a serious concern for an emerging anomaly vector. Furthermore, compromised SDN-enabled switches is greatly concerning as the visibility in SDN environment renders large, which might lead to a threat actor to gain a large attack surface to distinguish potential entities for targeting. One way to gain such an access privilege is through an exposure in the inspection procedure of our IDPS solution, KernelDetect, where a malicious payload could be misinterpreted [10].

In this work, we presume that the implementation of our KernelDetect is on secure kernel and it does not introduce any software or hardware bugs.

3. Related Work

String searching (a.k.a. string matching) is an applicable technique widely used in a variety of network security applications, such as IDS. The threat filtering and detection engine used in many popular IDS solutions such as Snort [19] and BRO [20] depend on early string searching algorithms like Bloom filter [40], Aho-Corasick [39], and Boyer-Moore algorithm [44] to identify network anomalies and threats. Up to today, there have been many studies to identify and examine each string matching approach [39, 40] for efficiency and robustness. However, these studies fall short in the efficiency of kernel-space detection. Furthermore, the study of IDS has been an important aspect of network security. There have been numerous developments of IDS solutions to deter malicious traffic [7, 8, 14, 15, 30, 33, 37, 42], but they heavily rely on user-space detection.

SDN boosts network traffic management and security as a security conflict can be feasibly resolved from the logically centralized control plane of SDN infrastructures. In this paper, we deploy SDN aspects to address a particular security challenge. Scott-Hayward, et al. [38] and Ahmad, et al. [48] summarized recent studies on the vulnerabilities of existing approaches in an SDN environment and security techniques that can strengthen the network-wide security in SDNs. Although some solutions such as FloodGuard [14], SPHINX [15], [46], and FortNOX [33] leveraged the SDN capabilities to provide anomaly detection and mitigation, they performance overheads are so significant that their applications are hindered.

SnortFlow [49] is a Snort-based [19] IDPS that leverages the strengths of Snort for pattern matching and content analysis to detect network intrusions in the cloud and deploy countermeasures in run-time. FortNox [33] is a security enforcement kernel deploying the NOX OpenFlow controller to address SDN tunneling attack and also check flow rule conflicts using a synchronous detection engine. Furthermore, Mahout [28] introduced a novel low-overhead framework for network traffic management that checks socket buffers to detect elephant flows and improve the prevention mechanism for flooding attacks in the SDN environment. SPHINX [15] proposes a flow graph-based prototype in an attempt to identify various threats in SDN traffic flows. RAID [16] is a SNORT-based IDS prototype that leverages the computational power of graphics cards and Aho-Corasick approach to offloading pattern matching computation, passively monitor the network, and to targeting operational exploitation in a large-scale environment. However, the effectiveness of this prototype was assessed only through OpenFlow backed connecting to three hardware switches. Moreover, Wang, et al. [35] proposed an entropy-based solution to address the problem of DoS attacks and monitor the detection accuracy of such a network threat. FRESCO [29] is a prototype developed within the application layer and the control layer of SDN. Its design facilitates the implementations of new security features in addition to prebuilt security modules that could be easily integrated in any SDN controller.

TopoGuard [23] is introduced to address the topological poisoning vulnerabilities and threats in SDN environments based on security omission's fixation. By introducing a practical approach based on an extension of NOS design, Rosemary [36] addressed the issue of control layer resilience that affects the control plane. However, these solutions have mainly focused on protecting the data plane of SDN controller from malicious applications, our lightweight kernel-level solution is capable to dynamically update the rule set in SDN environments and optimize traffic inspection while detecting network anomalies through dynamic switching between two string matching and filtering mechanisms.

Furthermore, VeriFlow [31] is a layer-based solution residing between SDN controller and network devices that dynamically checks for network-wide invariant violations during the insertion of forwarding rules. But, the proposed prototype is only examined through Mininet emulator [13]. NetPlumber [32] is a real-time policy verification framework that incrementally checks for compliance of state changes, using the Header Space Analysis (HSA) aspect. NICE [37] is a distributed multiphase-based framework deploying OpenFlow APIs to implement a monitoring module for network bugs and collaborative attack detection and mitigation in a cloud virtual networking environment. While FAST [6] identifies areas in conducting a forensic study on switching devices.

Most of existing studies deals with withstanding a single type of network anomalies using SDN capabilities and techniques, e.g., Wang, et al. [35]. SDNScanner [34] and AVANT-GUARD [30] introduces a practical solution based on connection migration techniques to solve the problem of scalable control plane saturation attack, by altering flow management at a switch level, but their approaches exposed only the flows that complete a TCP handshake based on a SYN proxy implementation and are limited to TCP saturation attacks only.

To the best of our knowledge, KernelDetect gives the first kernel-level solution instead of traditional user-space IDPS ones. It is a lightweight kernel-level detection mechanism. Contrary to the existing conventional work, we investigate IDPS on a kernel

space that overcomes the implementation difficulty of a kernel space (e.g., SoftFlow [4]). As Snort and Bro are popular tools in this area, we choose them in our comparison study.

The majority of existing solutions to building network security applications on top of SDNs are either not practical or limit performance and or deployability. While most existing evaluation methods deploying SDN for detection and mitigation are based on Mininet [13], a network emulator whose results might not be practical and far from real-world scenarios.

4. Threat Models and Attack Vectors

As stated before, SDN infrastructures are vulnerable to a variety of security attacks While SDN is widely used in different applications such as network traffic management. These various serious attacks include DoS [35], LDS [23], and man-in-the-middle attack (MITM) [23]. They have been found in an SDN environment. That is, launching these three types of attacks on SDN can be found in [23, 35]. Without loss of generality, we concentrate on DoS attacks in this paper, where we examine adverse users who may launch DoS attacks and normal users (or called clients) who launch a series of normal network traffic within an SDN environment. That is, the attack vector of this research is DoS and DoS [35] demonstrates its threat model in SDN. In this research, we will study the implementation method of DoS attacks in an SDN environment and periodically launch such attacks on GENI as described in Section 6, where we will examine the effectiveness of KernelDetect in terms of detection and mitigation time as well as its accuracy.

In this research, we assume that all SDN controllers and switching devices are safe from a threat actor, but leave end devices vulnerable to attacks. Mitigation is a critical way to thwart an attack and to prevent false positive events carefully; whitelisting will be required.

Whitelisting is a common approach to safeguarding mitigation faults such as disabling the WAN interface at an edge router and a network link to a known trusted computing device. In our threat detection approach, we do not implement any whitelisting for end devices attached to SDN switches as all users can be adverse at some point of time. Moreover, using KernelDetect, we implement detection on each suitable switching device for inspection purposes that will be further described in our experimental evaluation. Inter-switch links, commonly identified as a shared network link between two switching devices, contain a variety of network traffic intent from malicious to a normal user. Moreover, if these links were to be disabled through mitigation techniques, network operations would potentially fail. We inter-switch links to prevent mitigation faults from occurring. Although safeguarding inter-switch links provides reassurance from mitigation faults, a compromised end device has a greater potential to establish a significant threat to an SDN environment.

Lastly, we treat KernelDetect trustworthy even though adverse users can potentially obfuscate, exploit or overfill buffers specific to IDS solutions in addition to our string matching methods, Bloom filter, and the Aho-Corasick algorithm. We will identify an attack method in our experimental evaluation of Section 6. Following our evaluation, we have also investigated an IDS solution for other threats. However, we only present our study for DoS in this paper due to the page limit.

5. Design of KernelDetect

This section presents the architectural design of KernelDetect with discussions. We further discuss a threat signature structure for our proposed detection solution.

5.1. KernelDetect Placement and Architecture

The placement of KernelDetect is critical to detection and mitigation timings of an emerging threat. Before we present the architectural design of KernelDetect, Figure 1 shows the location and functionality of KernelDetect whose implementation is done in a configuration that operates in tandem with an SDN switching device.

Traffic duplication occurs within KernelDetect as both KernelDetect and OVS utilize raw socket communications in the back-end of the software system. The SDN Controller receives REST API calls from each switch when identifying a threat for mitigation. In this research, we use Floodlight as the controller software due to its REST API features. Figure 2 provides an architectural design of KernelDetect for both traffic inspection and signature matching with decision-making processes.

Let kds be a KernelDetect score, a an administrative-set incremental value for adverse traffic, b a decremental value for trustworthy traffic, and kdt a threshold value to determine whether such traffic should be placed in an inspection through either the Aho-Corasick algorithm or Bloom filter, called *Aho-Corasick inspection* or *Bloom filter inspection*, respectively. M simply denotes the matching scheme for KernelDetect.

When traffic enters an interface on a respective switch, the value is temporarily stored, and the information is forwarded to OVS and KernelDetect for their appropriate purposes of forwarding and inspecting traffic, respectively. During the initial state of KernelDetect, that is, when the service begins, an administrative configuration is examined to verify if a secure mode is enabled. We define the secure mode as a parameter such that if the placement of the switching device is in a critical data region, KernelDetect will

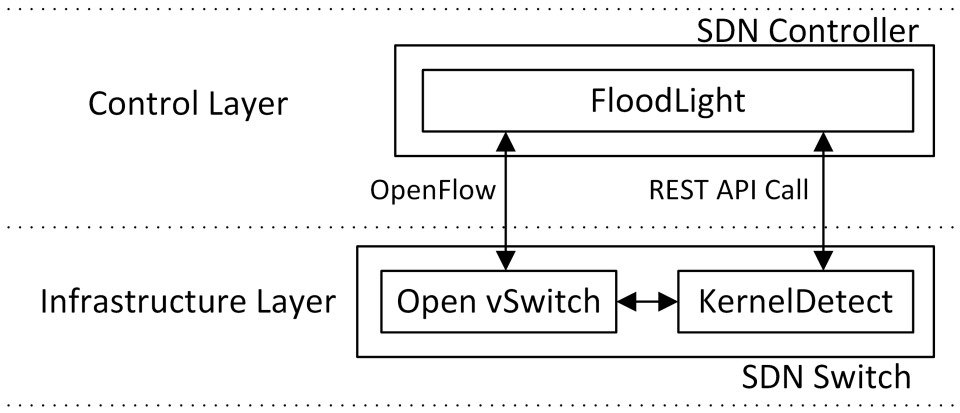

Figure 1. The placement and functionality of KernelDetect for network traffic flows.

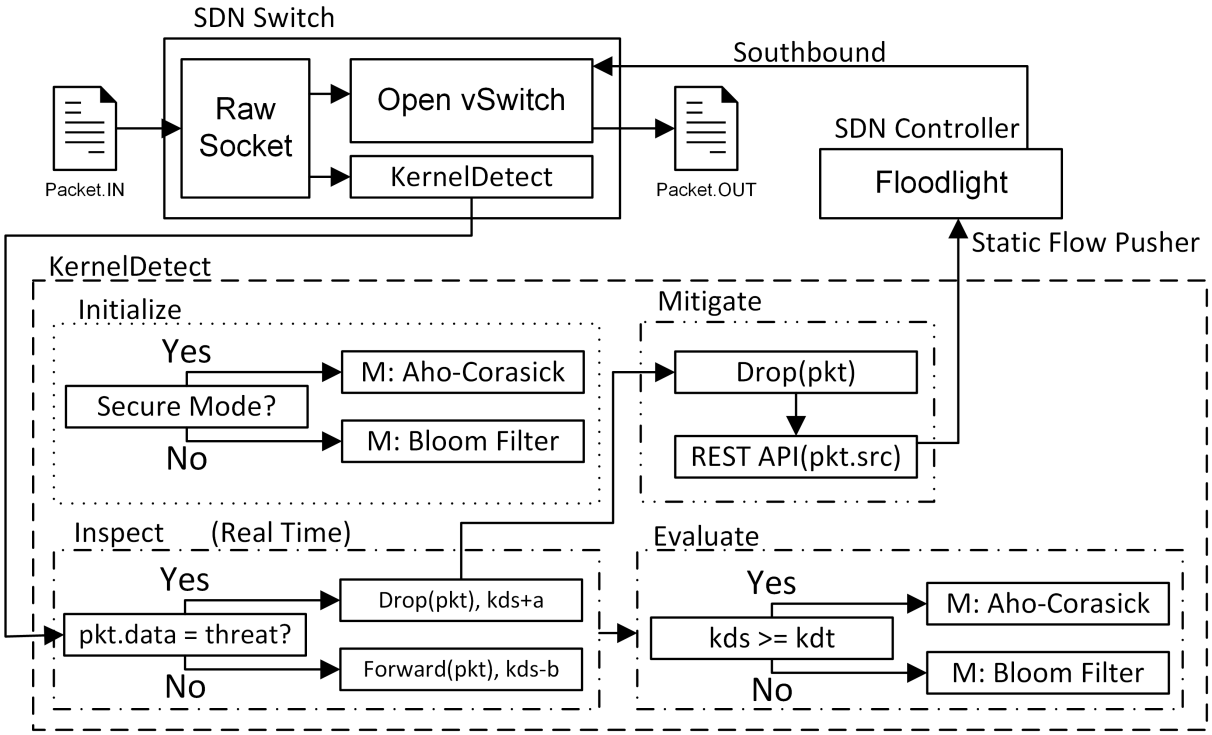

Figure 2. The architecture of KernelDetect consisting of four states: "Initialize," the beginning state of the SDN switch operations, "Inspection," a real-time inspection of traffic obtained from the raw socket of the operating system, "Mitigation," a critical step to thwart an attack and to prevent false positive events carefully, and "Evaluation," the examination of incoming traffic through 'Aho-Corasick' or 'Bloom filter' with a global view of the network.

enforce a detailed inspection using Aho-Corasick. If the placement does not have severe inspection approaches, then KernelDetect may use Bloom filter for detection. During the inspection process, we identify and examine to see whether the traffic has malicious intent through signature matching. If the intent is considered trustworthy, then we simply forward the traffic and decrease kds by a value of b, and add a when the intent is not trustworthy. Using a threshold condition of comparing kds to kdt, we examine whether future traffic should remain in Aho-Corasick or Bloom filter inspection. If the traffic has a malicious intent, we simply drop the packet from the raw socket and inform the SDN controller using REST API calls to block the adverse threat.

5.2. Threat Signature Structure

Identifying adverse network traffic could be challenging as it depends on IDS signatures and threat identification markings. Particularly, two common approaches

are considered to identify traffic threat through string-based matching, and traffic over time where an observation of a pattern of network packets occurs in a given period. As mentioned before, although KernelDetect applies to various attacks, this paper focuses on a DoS attack vector due to the page limit.

DoS: The identification of a DoS attack can be a challenge in an at-scale network. There are multiple methods to create a DoS attack from TCP SYN-flooding to other detailed approaches such as OSI Layer 7-based flooding. Like [35], we can identify a traffic pattern over an interval of time to determine if there is a DoS attack. That is, if the quantity of traffic exceeds a given threshold, KernelDetect considers that a DoS attack occurs, and it raises an alert. This threshold is a fixed value among all the approaches studied in our experiments later. The correlation with signature matching relates towards the frequency of alerts that is, KernelDetect raises an alert when a match occurs. The observation of a threat can originate from one or multiple sources where the attacker may spoof the source address of the DoS. Based on this given knowledge, KernelDetect accounts for such threats.

Signature-based matching may not be the appropriate tool to detect DoS attacks where the adversary can often insert arbitrary data into a packet payload. This approach renders signature-based detection ineffective. In some cases, DoS attacks may not have any form of data for its payload, such as a low-profile TCP SYN flood attack. However, KernelDetect, considers matching the header information of a network packet rather than its packet payload, which increases the performance of threat detection. Below is the algorithm for KernelDetect where TH and THP are threshold values for time and packet intervals, respectively.

```
P = PACKET_IN
while P do
  TS = TIMESTAMP
  if P.TYPE == ICMP then
    Q{P.SRC_ADDR}++
    if P.SRC_ADDR NOT IN S then
      S{P.SRC_ADDR} = TS
    else
      if TS - S{P.SRC_ADDR} > TH then
        if Q{P.SRC_ADDR} > THP then
          REST API Call to SDN Controller
        else
          S{P.SRC_ADDR} = TS
        end if
      end if
    end if
  end if
end while
```

6. Experimental Evaluation

This section presents our experimental design and evaluation of KernelDetect. In order to conduct a comprehensive examination of KernelDetect, we manipulate different experimental parameters such as the varying number of transmitted packets, threshold time, and the duration of experiments.

6.1. Experimental Topology Design

To measure the effectiveness of KernelDetect, we utilize GENI [18] for experimental evaluation. GENI is a real-world heterogeneous virtual testbed with networking capabilities including SDN. To evaluate KernelDetect, we construct a topology with the following three constraints: (1) An adverse user attached to a single network link identifying major areas of mitigation. (2) A shared network link used by both a normal user and an attacker. (3) An edge network link that carries both normal and attack traffic. This edge link has limited SDN controller management. Figure 3 gives a visual view of the experimental topology that considers the previous research challenges where the locations of adverse users are explicitly labeled. For presentation purpose in this paper, we give a relatively simple topology for our evaluation as shown in Figure 3. However, KernelDetect is applicable to any complex network topology.

At a certain point of time, clients may be adverse or potentially compromised. However, we do not take into account this particular scenario in our evaluation experiments as we do not implement any whitelisting technique for end devices attached to SDN switches in our threat detection approach. For this reason, normal users could have the potential to be prone to mitigation techniques depending on IDS signatures and rule sets.

6.2. Detection Rules in BRO and SNORT

As SNORT and BRO are rule-based detection approaches, they are involved in an attempt to define a set of rules for the understanding of network traffic behaviors to determine whether or not the network traffic is adverse, SYN flood detection and mitigation rules needs to be manually downloaded and added to the rule set files in both BRO and SNORT accordingly, unless they were by default installed along with the detection systems. The following figure depicts the detection rule of SYN flood attacks (i.e. DoS) for SNORT IDS. In this rule definition, an alert is raised whenever 70 or more packets are sent within a 10 second interval that is TCP-SYN flagged.

```
alert tcp any any -> $HOME_NET 80(flags:S;
msg:"Possible TCP DoS is Detected !!";
flow: stateless; detection_filter: track by_dist, count 70,
seconds 10; sid 10001;rev:1;)
%\label{SYN-flood}
```

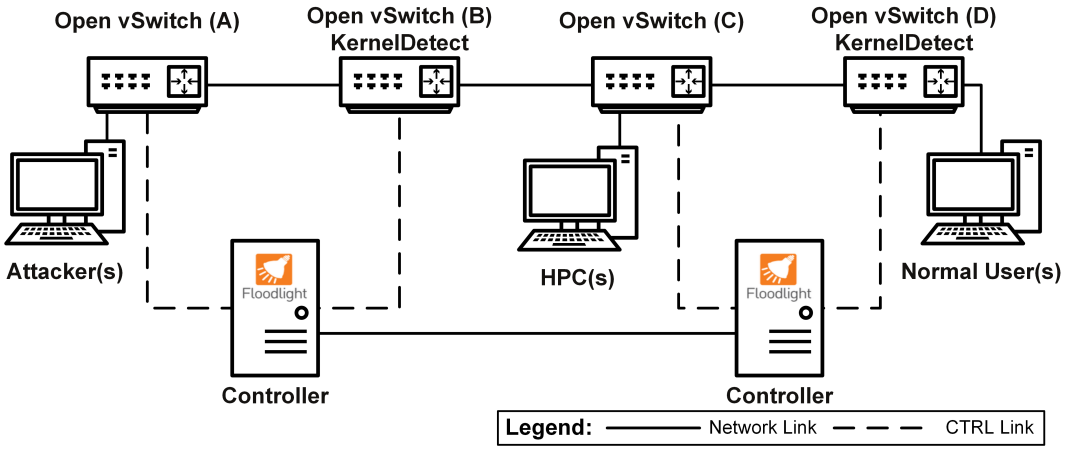

Figure 3. GENI experimental topology for evaluation where KernelDetect is only implemented in switches B and D as depicted in the diagram. Moreover, CTRL links are the communication medium between each SDN switch and their respective controller. Lastly, our experimental evaluation interchanges KernelDetect-enabled switches with SNORT and BRO for our comprehensive study.

6.3. Traffic Generation Techniques

Network traffic is comprised of two intent-types: normal and adverse. While adverse traffic is transmitted into our proposed solution, normal traffic will be mixed into our experiments to simulate a real-world environment. In order to provide hybrid network communication and mix normal traffic into the grand scheme of our experiments, we utilize D-ITG [21] and IPerf [22] as traffic generation software. Specifically, we utilize a culmination of both tools and increase the quantity of traffic for our experiments. Although we cannot fully emulate a normal user, we believe that iPerf tool should provide a fundamental approach to measuring our solution as it provides the ability to saturate a network link in addition to real-time network throughput analysis. To be concise, we configure iPerf with the default parameters for operational use.

6.4. Experimental Results

The performance of an IDS/IPS solution is critical to counter adverse network threats and, specifically, threat actors. As traffic flows from one host to another, congestion, computational bottlenecks and system's resource draining can occur within a network environment and in regards to an IDS solution, packet inspection and the level of detail can produce resource strain on a computing device. In this section, we evaluate the performance of our KernelDetect in term of its inspection time, mitigation time, detection accuracy, and system resource consumption comparatively compared to SNORT and BRO.

Inspection Time. To combat network anomalies, it is important to consider run-time filters as an independent threat detector able to detect attacks against itself. Such threat detection, together with judicious rate-limiting of traffic forwarded to full packet inspection, allows the detection and mitigation of threats while preserving the overall improvements in NIDS performance, including but not limited to, the average inspection time. For this reason, a full packet inspection procedure that is critical to the mitigation of threat actors and adverse network traffic need to be performed whenever an IDS/IPS is placed in service. Specifically, as packets arrive at an IDS, the information is placed in a buffer and waits for inspection. This waiting time increases the time needed to mitigate the adverse threat if the packet has malicious intent. To measure such inspection time, we established a near-equal configuration for each IDS solution with similar parameters of threat signatures in each respective database for measurement fairness purposes. We establish this approach to examining the effectiveness of each solution to have near mirror-like configurations and to examine the performance of each IDS solution closely.

To measure inspection time, we establish the communication between two devices using hping3 where we transmit small-size packets with the large quantity of traffic at one nanosecond interval of time, achieving a link saturation. We evaluate three threshold values of 5, 10, and 15 seconds for detection. In this evaluation, we run all the experiments 10 times and then average their results. Figures 4, 5, and 6 show the average inspection time for the threshold of 5-seconds, 10-seconds and 15-seconds, respectively. Our experimental results demonstrate that KernelDetect has the overall lowest average inspection time compared to SNORT and BRO.

Table 1 demonstrates that KernelDetect has lower inspection time average comparatively to BRO and SNORT while Table 2 describes a 95% confidence

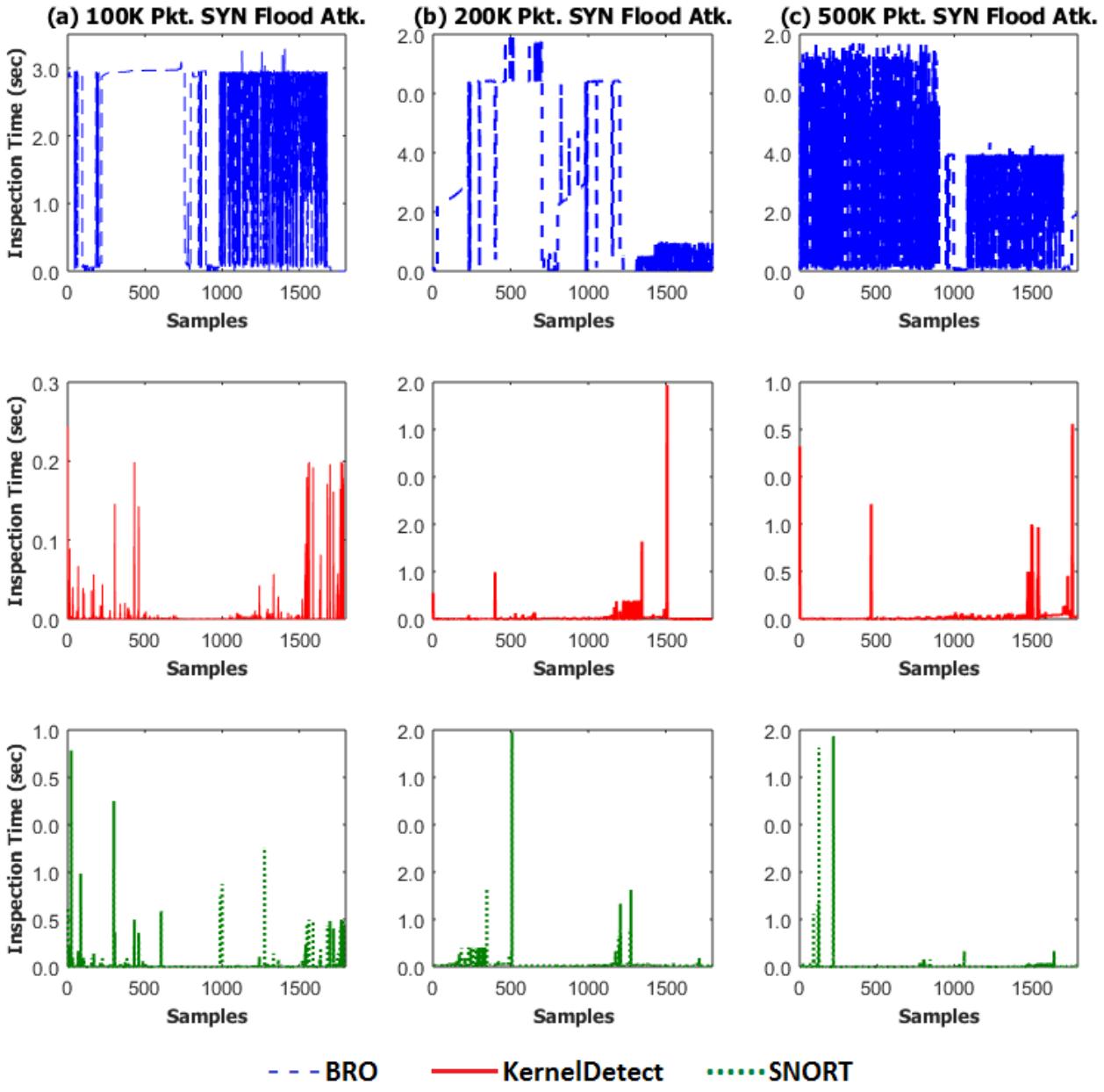

Figure 4. A comparative analysis of the inspection time for each IDS under 100K, 200K and 500K SYN flagged packet DoS attack using a 5 second threshold signature.

interval statistic. In Table 2, we only compare KernelDetect with SNORT because BRO has much higher inspection time than KernelDetect and SNORT as shown in Table 1 so that it would not be helpful even if we included a 95% confidence interval statistic for BRO in the table. Furthermore, although KernelDetect has some confidence interval overlap with SNORT, it is demonstrated in Table 2 that KernelDetect is still a clear winner in comparison to SNORT under various traffic loads with different signature threshold values regarding inspection time.

Mitigation Time:. In an IDS/IPS solution, mitigation time (a.k.a. time to mitigation) is the interval of time from when the first alert is raised once malignant network packets are captured at IDS/IPS doorstep to the time when an action is taken to combat them or until the threat is stopped. This time to mitigate is likewise when the infrastructure of an SDN environment is at its most vulnerable as the longer the delay, the more likely the attack will be successful. As this mitigation delay relies on the processing abilities of the IDS/IPS and its filtering mechanisms, it is key to ensuring the safety and well-being of a

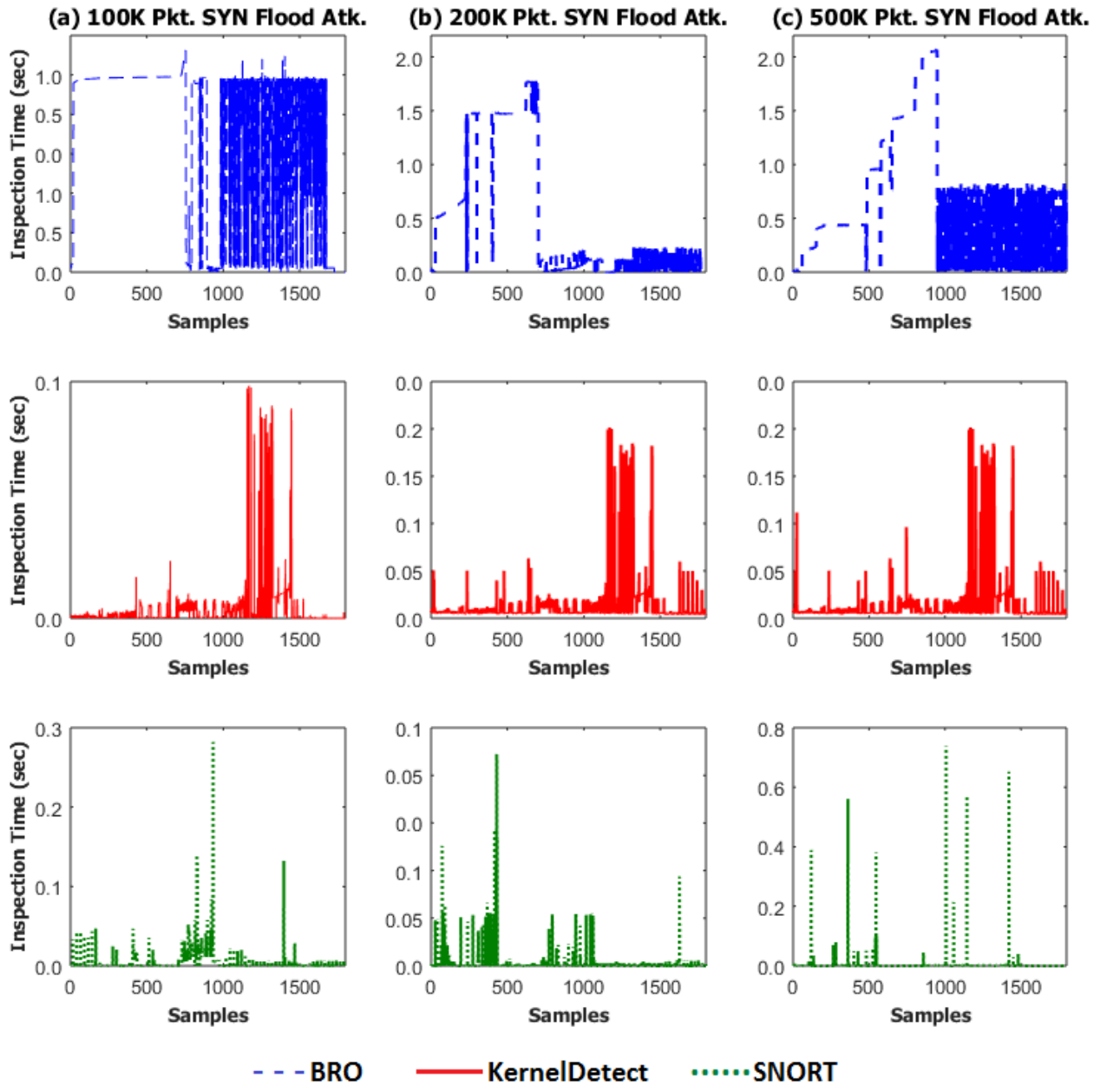

Figure 5. A comparative analysis of the inspection time for each IDS under 100K, 200K and 500K SYN flagged packet DoS attack using a 10 second threshold signature.

network at scale. Although each IDS has its unique attributes to capture an adverse attacks and mitigate them, we comprehensively examine the efficiency and robustness of KernelDetect solution by measuring its threat mitigation.

To measure the mitigation time, we examine the time between the initiation of each network attack and compared it to the time needed to rectify the threat as shown in Figures 7, 8, and 9, which is also represented in Table 3 as an average mitigation delay, and expressed using a 95% confidence interval in Table 4. Although Table 3 demonstrates that KernelDetect has a nearly-similar mitigation time as SNORT, it is still slightly better than SNORT in term of average, and significantly superior than BRO. Furthermore, similar to Table 2, we excluded BRO from Table 4 for the same reason discussed in inspection time section. As expressed in Table 4, KernelDetect solution shows better mitigation performance than SNORT when comparing their corresponding confidence intervals. Figures 7, 8, and 9 depict series of DoS attacks executed in the SDN environment using 5-seconds, 10-seconds and, 15-seconds threshold respectively where each solution

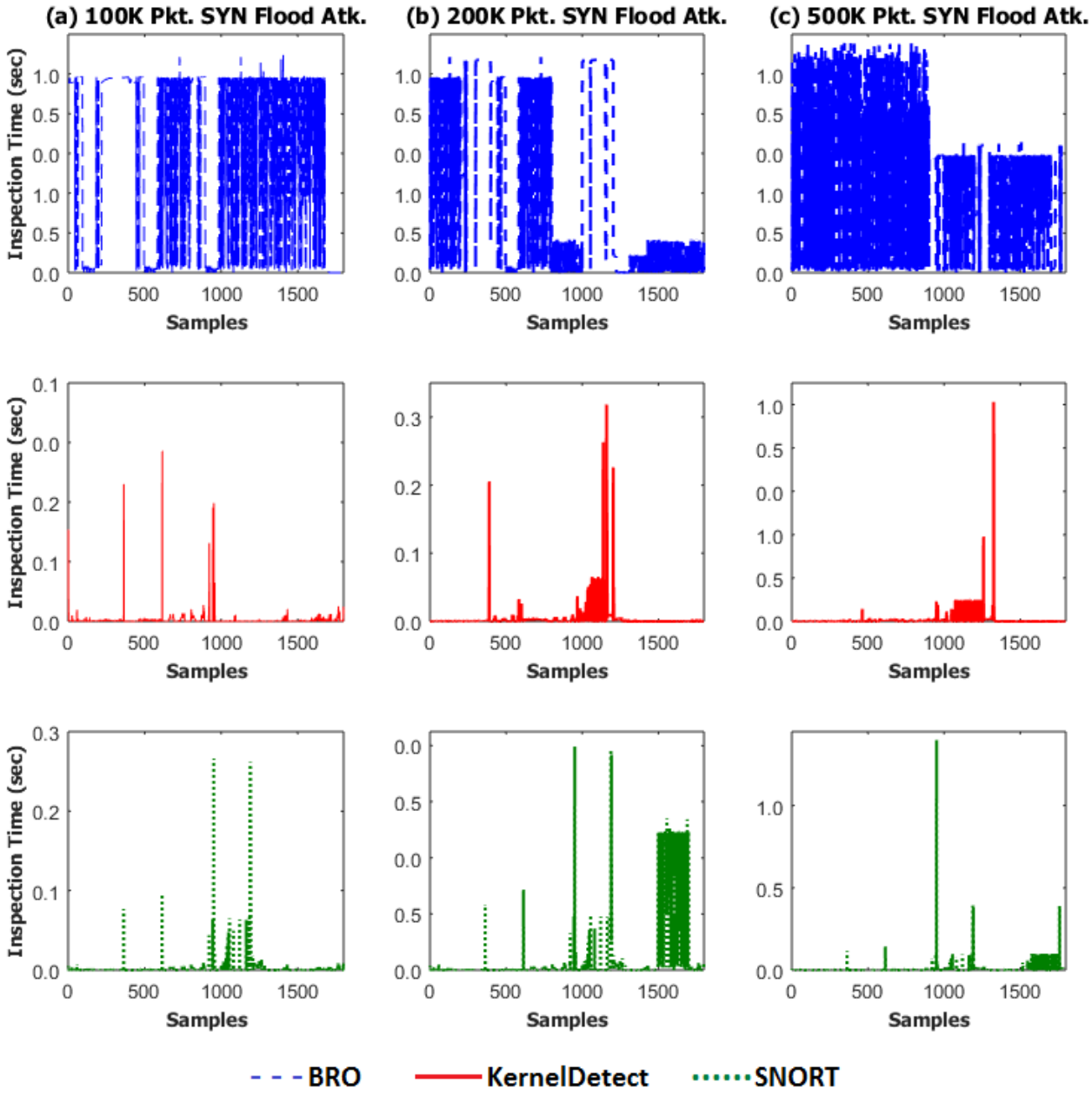

Figure 6. comparative analysis of the inspection time for each IDS under 100K, 200K and 500K SYN flagged packet DoS attack using a 15 second threshold signature.

provided necessary alerting and mitigation procedures. The mitigation technique for each solution utilizes the same function such that when an alarm is raised, the message presented will be used to block the respective address. Figure 10 presents the clearness of mitigation time using 10-seconds threshold value, and therefore proves that KernelDetect is a better solution to guarantee small mitigation delay.

False Negative and False Positive. False positive state is when the IDS characterizes a normal network traffic as a security threat. False negative state is when the IDS characterizes a threatening network traffic as an acceptable behavior. That is, a false negative is when the IDS fails to detect an attack and raise the threat alarm. Hence, a false positive and false negative can lead to significant downfalls in security of network communication. Figure 11 depicts the use of Receiver Operating Characteristic (ROC) curve methods to demonstrate the false positive rates for KernelDetect, SNORT, and BRO. Notably, the curve demonstrates our detection matching sensitivity for our experimental evaluation where we identify the accuracy of each system. BRO demonstrated to have the poorest accuracy

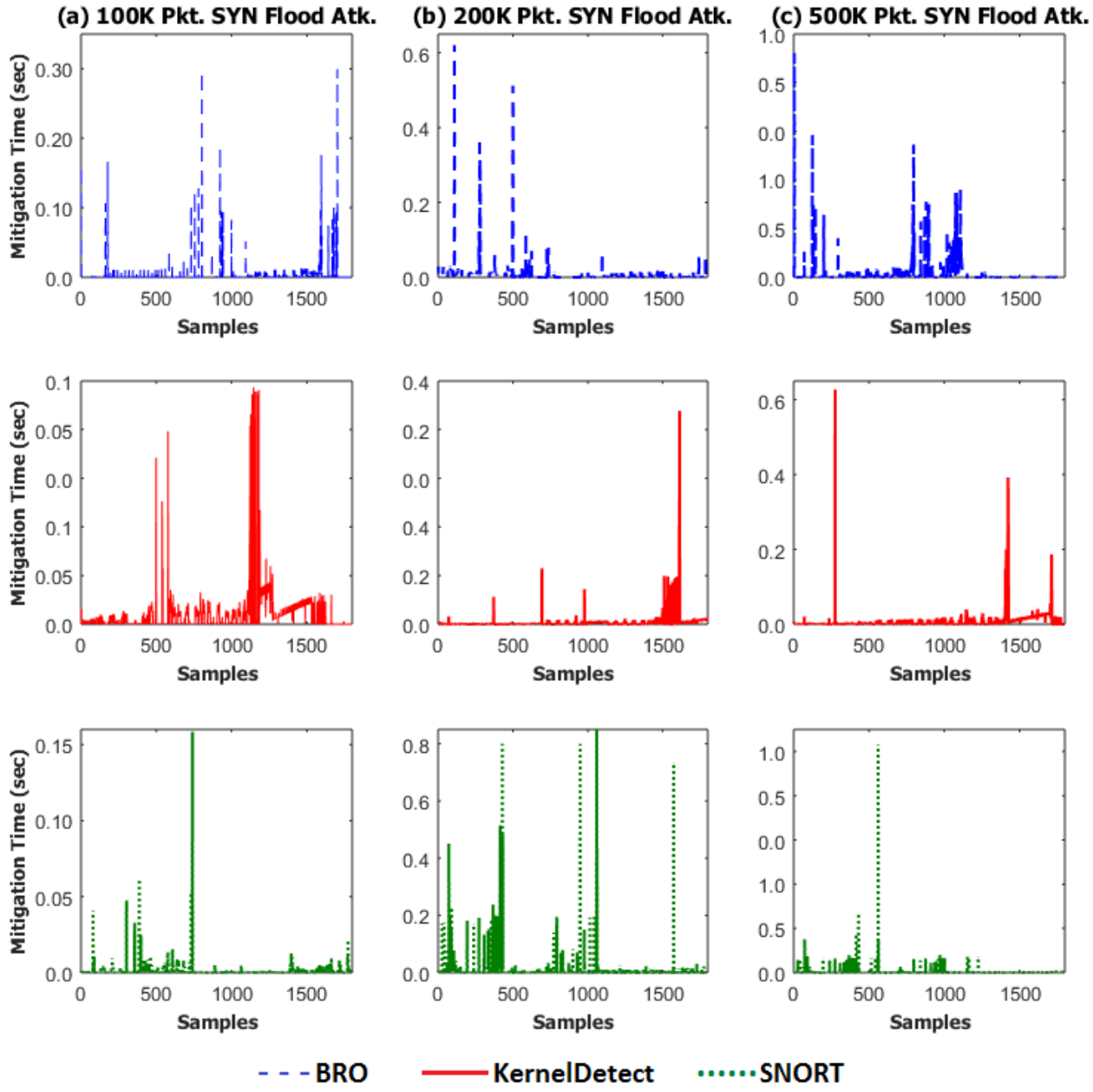

Figure 7. Threat mitigation time for each IDS under 100K, 200K and 500K SYN flagged packet DoS attack using a 5 second threshold.

rate in comparison to KernelDetect and SNORT where KernelDetect presented the most accurate results based on the analysis of the experimental results.

System Resource Utilization. The performance of an IDS/IPS solution is critical to counter adverse network threats and specifically—threat actors. As traffic flows from one host to another, congestion and computational bottlenecks can occur within a network environment in addition to an IDS solution. Inspection and the level of detail in examining the content of the packet can produce resource strain on a computing device. As network packets traverse between two devices, the information is stored in a buffer, waiting for inspection and forwarding purposes. Hence, memory resource consumption is vital in the operation of an SDN device and a critical segment for resource examination. Figure 13 depicts the performance of resources usage in term of average memory consumption for each IDS under 500K SYN flagged packet DoS attack using a 10-seconds threshold signature. In comparison with SNORT and BRO IDSs, the obtained results demonstrate that KernelDetect consumes less system resources when idle and under attack. Memory utilization increases during the events of a DoS attack

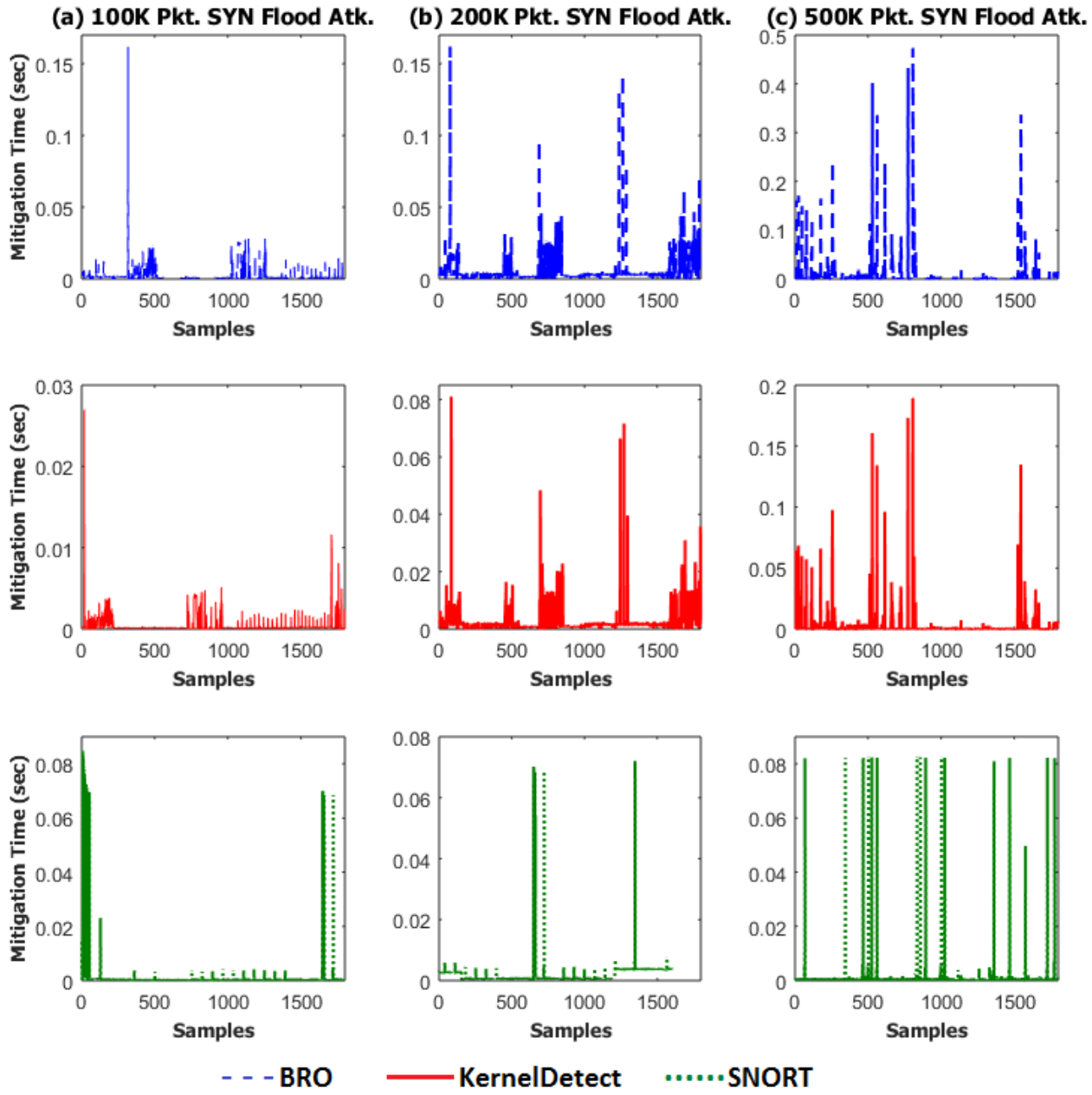

Figure 8. Threat mitigation time for each IDS under 100K, 200K and 500K SYN flagged packet DoS attack using a 10 second threshold.

where KernelDetect is more efficient than SNORT and BRO in the events of a post-DoS scenario. To be concise, once the DoS attack ends, both BRO and SNORT maintain constant memory resource utilization while KernelDetect's usage reduces to the lowest percentage rate. Moreover, to measure averaging CPU utilization in a kernel space, samples of system resource utilization are used where each IDS is under 500K SYN flagged packet DoS attack using a 10-seconds threshold signature. Figure 12 demonstrates that BRO has a higher-level system resource utilization in comparison to KernelDetect and Snort and KernelDetect shows significantly less average load in term of CPU usage.

Discussions. A kernel panic is one serious challenge in the use of kernel-space detection for a security apparatus such as KernelDetect. Moreover, if a kernel panic would occur to an SDN device, practical and operational usage would be lost. Additionally, the library functions that are implemented and imported into the design of KernelDetect may propose a vulnerability that could be haphazardous to the SDN environment. In the design of KernelDetect, this research treats all utilized libraries as trusted modules

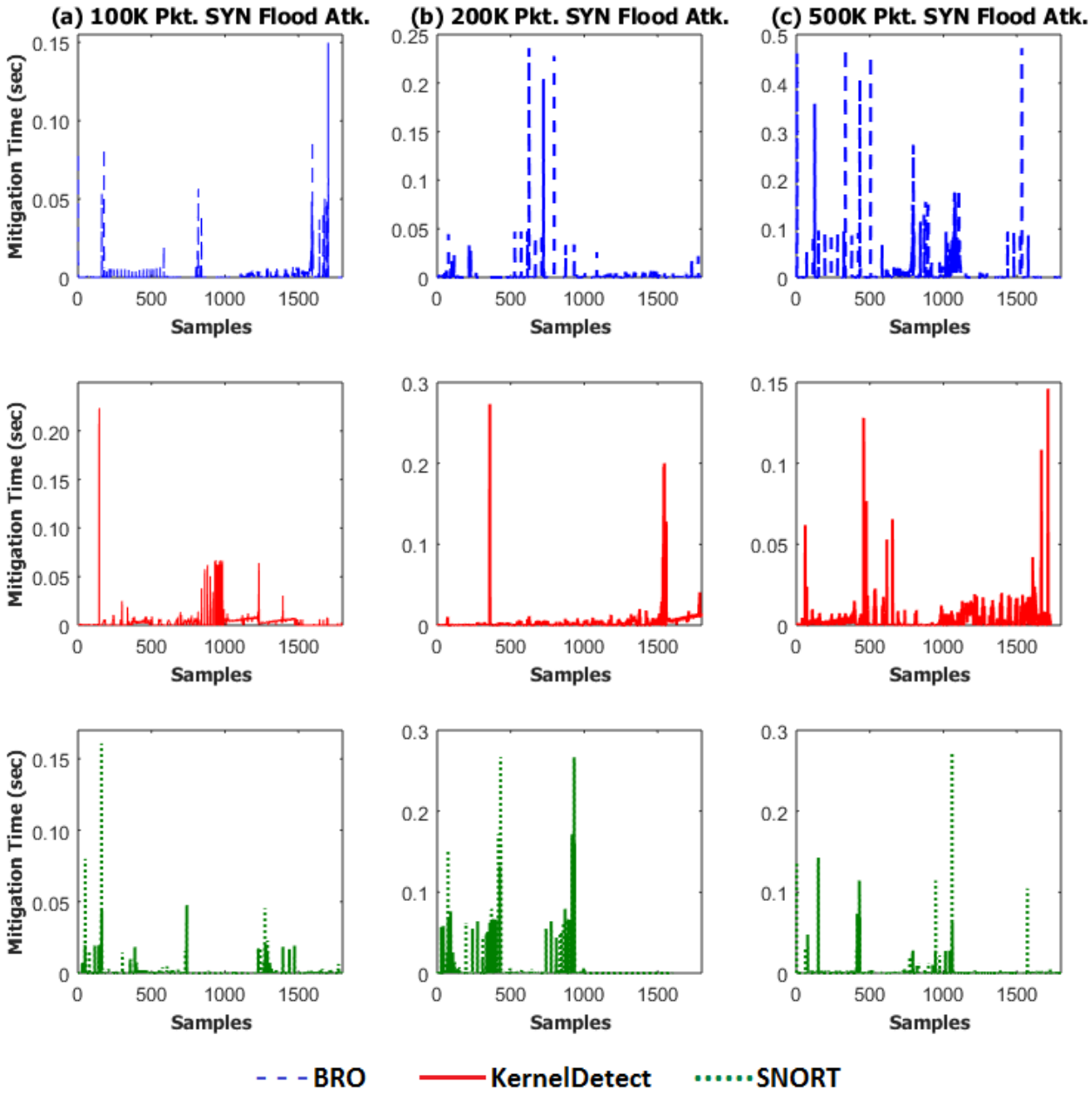

Figure 9. Threat mitigation time for each IDS under 100K, 200K and 500K SYN flagged packet DoS attack using a 15 second threshold.

in the implementation such that the discovery of a serious vulnerability would be well-known and urgent for patching purposes. One configuration that may be sub-optimal for an SDN environment is to implement KernelDetect on an independent computing system that is attached to a port mirroring interface using a SPAN/TAP configuration such that if kernel panic would occur, SDN switching operations would continue to function. Lastly, KernelDetect utilizes raw socket information to read incoming packets. This read procedure could be insufficient for the switching operation such that OVS could process the raw socket information at a faster rate than KernelDetect.We experimented by creating a small packet size (i.e. 70 bytes as packet body size) full link saturation scenario, but we were unable to emulate the concern. Our belief to such an event would potentially be plausible in a large network throughput interfaces such as 100Gbps. However, our evaluation was limited to only 1 Gbps speeds. Performance modeling like [43] is helpful to such studies.

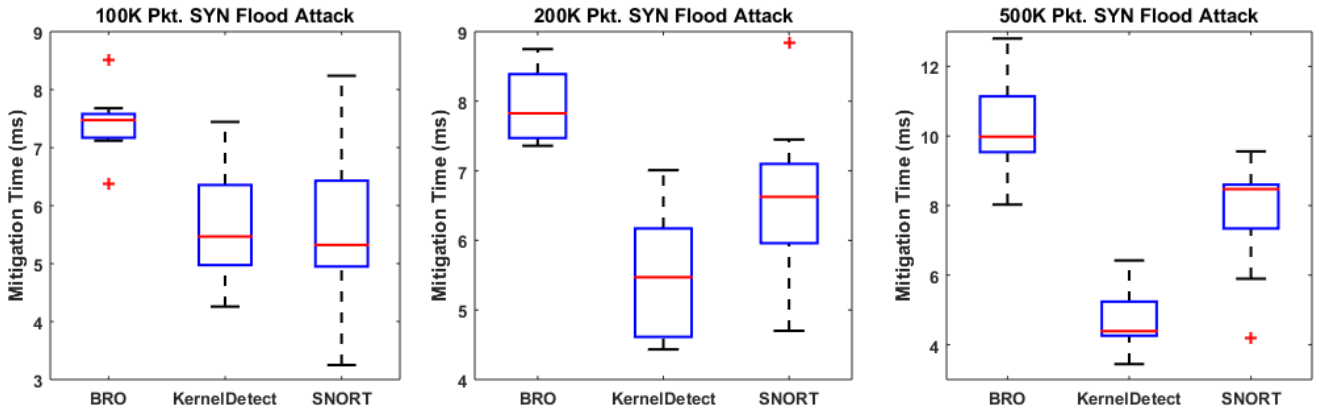

Figure 10. Threat mitigation for each IDS under 100K, 200K, and 500K SYN flagged packet flood attack using 10 second threshold detection technique and represented as a box plot.

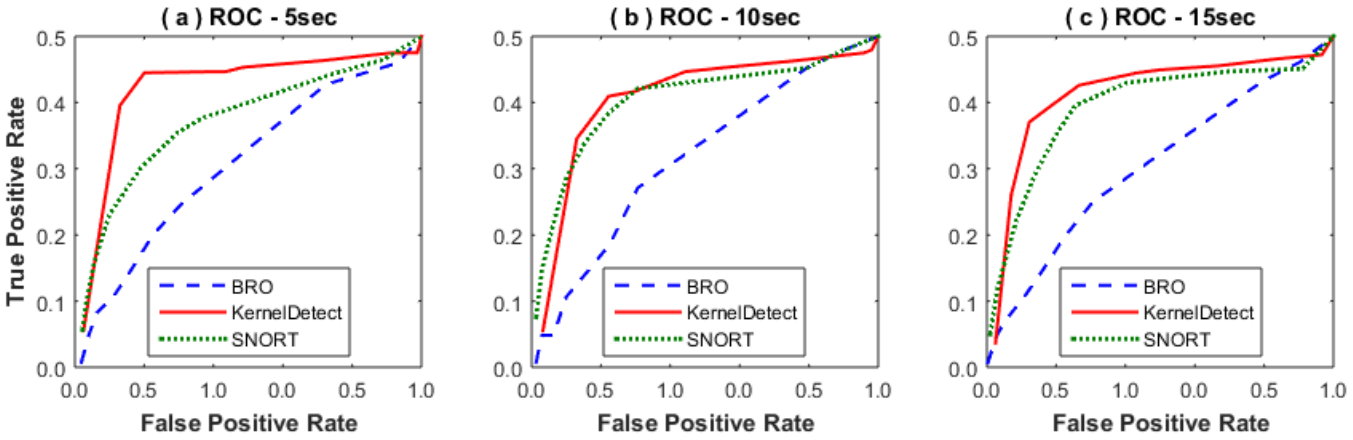

Figure 11. ROC Curve for threat detection for each IDS under 100K Packets SYN flood attack using various thresholds of 5, 10 and 15 seconds for threat signatures of a DoS attack.

Table 1. A comparison of the average inspection time in seconds among KernelDetect, SNORT and BRO under various traffic loads of 100K, 200K, and 500K SYN flagged packets using detection thresholds of 5, 10, and 15 seconds.

Traffic Load (K)		100	200	500
Threshold (Sec.)	IDS			
5	KernelDetect	0.0048	0.0047	0.0109
	SNORT	0.0033	0.0186	0.0319
	BRO	2.1264	1.5187	2.3996
10	KernelDetect	0.0106	0.0111	0.0112
	SNORT	0.0128	0.2686	0.0643
	BRO	2.2656	1.1270	4.3337
15	KernelDetect	0.0067	0.0070	0.0069
	SNORT	0.0067	0.0243	0.0172
	BRO	1.9113	1.3433	2.5786

7. Conclusions and Future Work

In this paper, we have introduced KernelDetect, a lightweight kernel-level intrusion detection and prevention framework. KernelDetect is a modular countermeasure framework in an SDN environment. It leverages modular string searching and filtering mechanisms along with SDN techniques and features. KernelDetect is designed in a way that leverages the strengths of both Aho-Corasick and Bloom filter with SDN controller techniques. Although KernelDetect approach can be applicable to handle and thwart different types of network anomalies and adverse threats, in this paper, we only explore and focus on the events of a DoS attack in an SDN environment.

We have exploited and dynamically integrated modular detection mechanisms that are an exact string matching and a partial matching algorithms together to thwart the above network threats that reside in the application plane of an SDN environment.

In order to test the validity of our KernelDetect IDPS and evaluate its performance, we conducted comprehensive evaluation experiments on GENI, a real-world testbed infrastructure where we manipulated the experimental parameters of SYN flood attacks,

Table 2. 95% confidence interval statistic for inspection time (seconds) between KernelDetect and SNORT where L and U represent their lower and upper bound, respectively.

Traffic Load (K)		100			200			500		
Threshold (Sec.)	IDS	Stdev	L	U	Stdev	L	U	Stdev	L	U
5	KernelDetect	0.0142	0.0024	0.0031	0.0025	0.0037	0.0056	0.2009	0.0066	0.0150
	SNORT	0.0229	0.0027	0.0039	0.0624	0.0162	0.0211	0.3644	0.0170	0.0470
10	KernelDetect	0.1711	0.0038	0.0174	0.0183	0.0103	0.0119	0.0186	0.0104	0.0120
	SNORT	0.2309	0.0082	0.0173	0.2217	0.2549	0.2824	0.2566	0.0592	0.0690
15	KernelDetect	0.0522	0.0037	0.0068	0.0602	0.0051	0.0088	0.0654	0.0055	0.0080
	SNORT	0.0558	0.0050	0.0083	0.0984	0.0195	0.0253	0.1205	0.0137	0.0210

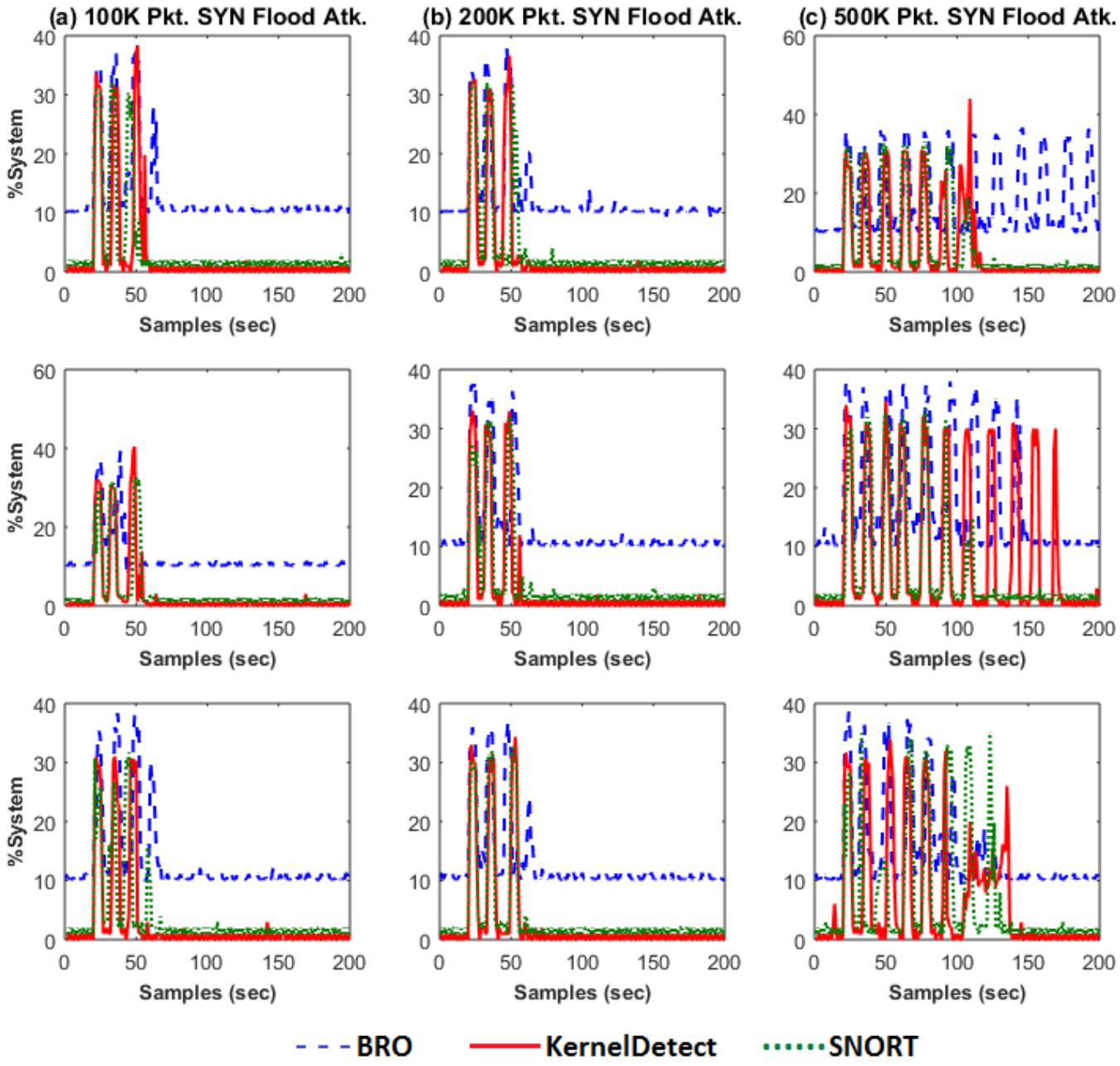

Figure 12. System usage at 100K, 200K, and 500K packets (Pkt.) loads of SYN flood attack (Atk.)

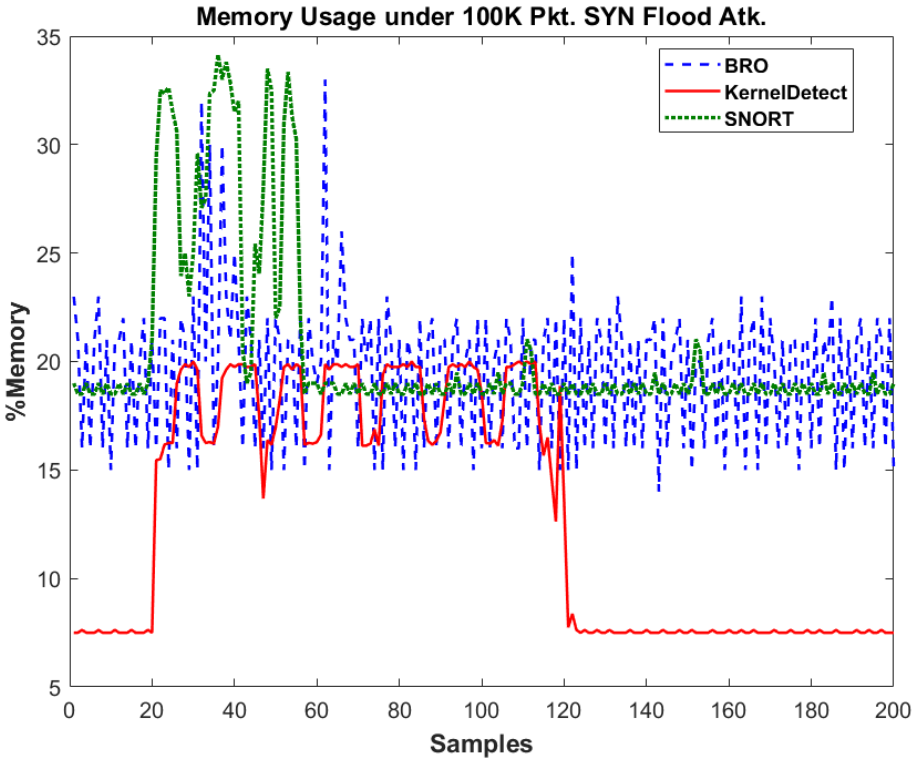

Figure 13. Memory utilization for each IDS under the scenario of a single attack with 100K SYN flagged packets (Pkt.) using a 10 second threshold detection rate. Additionally, a full-link saturation is achieved during the attack in this evaluation.

Table 3. The average mitigation time in seconds for KernelDetect, SNORT and BRO under various traffic loads of 100K, 200K, and 500K SYN flagged packets using detection thresholds of 5, 10, and 15 seconds.

Traffic Load (K)		100	200	500
Threshold (Sec.)	IDS			
5	KernelDetect	0.0036	0.0074	0.0123
	SNORT	0.0056	0.0012	0.0130
	BRO	0.0060	0.0100	0.0108
10	KernelDetect	0.0060	0.0054	0.0044
	SNORT	0.0055	0.0065	0.0078
	BRO	0.0074	0.0080	0.0118
15	KernelDetect	0.0042	0.0049	0.0086
	SNORT	0.0071	0.0101	0.0200
	BRO	0.0096	0.0635	0.1254

including but not limited to, the number and length of network packets (the number of network packets ranges from 100K to 500K and launched SYN flooding attacks with different packet sizes) and sampling times. In order to compare the effectiveness and validity of KernelDetect with traditional IDSs, SNORT [19] and BRO [20], we have measured the performance constraints and overhead, utilization of system resources, average inspection and mitigation time, true positive, false positive, and false negative for each IDS among 10-run experiments in order to identify and address network scalability where traffic quantity can be saturated for an SDN environment.

Through our extensive evaluation for each IDS performance, We have shown that our KernelDetect solution is an effective and efficient approach to detect and thwart the DoS threats within the data plane of an SDN environment.

In our future work, we plan to extend and adjust the implementation of our KernelDetect solution to deal with different types of network anomalies and adverse threats such as man-in-the-middle (MITM) attacks and Link Discovery Service Exploitation (LDS). Moreover, we plan to identify the potential areas of threat detection in control plane communications.

Acknowledgement. We acknowledge National Science Foundation (NSF) to partially sponsor the work under grants #1633978, #1620871, #1620862, and #1636622, and BBN/GPO project #1936 through NSF/CNS grant. We also thank the Florida Center for Cybersecurity for a seed grant. The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies, either expressed or implied of NSF.

References

[1] Xiong K., Wang R., Du W and Ning P. (2012) *Containing bogus packet insertion attacks for broadcast*

Table 4. 95% confidence interval measurements for mitigation time (seconds) between KernelDetect and SNORT where L and U represent their lower and upper bound, respectively.

Traffic Load (K)		100			200			500		
Threshold (Sec.)	IDS	Stdev	L	U	Stdev	L	U	Stdev	L	U
5	KernelDetect	0.0141	0.0032	0.0040	0.0376	0.0063	0.0087	0.0966	0.0097	0.0149
	SNORT	0.0424	0.0044	0.0067	0.1137	0.0078	0.0167	0.0820	0.0094	0.0166
10	KernelDetect	0.0302	0.0051	0.0068	0.0355	0.0045	0.0063	0.0230	0.0038	0.0050
	SNORT	0.0542	0.0027	0.0037	0.0530	0.0026	0.0037	0.0190	0.0081	0.0093
15	KernelDetect	0.0267	0.0035	0.0060	0.0656	0.0030	0.0067	0.0629	0.0068	0.0104
	SNORT	0.0494	0.0058	0.0085	0.0921	0.0065	0.0137	0.0932	0.0096	0.0240

authentication in sensor networks. ACM Transactions on Sensor Networks (TOSN). Vol. 8. P: 20.

[2] Ko C., Fraser T., Badger L. and Kilpatrickv D. (2000) *Detecting and Countering System Intrusions Using Software Wrappers.* USENIX Security Symposium. pp: 1157-1168.

[3] Vasiliadis G., Antonatos S., Polychronakis M., Markatos E. and Ioannidis S. (2008) *Gnort: High performance network intrusion detection using graphics processors.* Recent Advances in Intrusion Detection (RAID). pp: 116-34.

[4] Jackson E. J, Walls M., Panda A., Pettit J., Pfaff B., Rajahalme J., Koponen T. and Shenker S. (2016) *SoftFlow: A Middlebox Architecture for Open vSwitch.* USENIX Annual Technical Conference. pp: 15-28.

[5] Chin T., Xiong K. and Rahouti M. (2017) *End-to-End Delay Minimization Approaches Using Software-Defined Networking.* Proceedings of ACM RACS 2017.

[6] Chin T. and Xiong K. (2017) *A Forensic Methodology for Software-Defined Network Switches.* Proceedings of IFIP International Conference on Digital Forensics. pp: 97-110.

[7] Chin T. and Xiong K. (2016) *Dynamic generation containment systems (DGCS): A Moving Target Defense approach.* Proceedings of 3rd IEEE International Workshop on Emerging Ideas and Trends in Engineering of Cyber-Physical Systems (EITEC). pp: 11-16.

[8] Chin T., Mountrouidou X., Li X. and Xiong K. (2015) *An SDN-supported collaborative approach for DDoS flooding detection and containment.* Proceedings of IEEE Military Communications Conference (MILCOM). pp: 659-664. Proceedings of 35th IEEE International Conference on Distributed Computing Systems Workshops (ICDCSW). pp: 95-99.

[9] Akella A. V and Xiong K. (2014) *Quality of service (QoS)-guaranteed network resource allocation via software defined networking (SDN).* Proceedings of IEEE International Conference on Dependable, Autonomic and Secure Computing (DASC). pp: 413-424.

[10] *Apache Spam Assassin Public Corpus.* https://spamassassin.apache.org/publiccorpus/

[11] Pfaff B., Pettit J., Koponen T., Jackson E. J, Zhou A., Rajahalme J., Gross J., Wang A., Stringer J., Shelar P. and others. (2015) *The design and implementation of OpenvSwitch.* Proceedings of the USENIX Symposium on Networked Systems Design and Implementation (NSDI). pp: 117-130.

[12] Ahrenholz J., Danilov C., Henderson T. R and Kim J. H (2008) *CORE: A real-time network emulator* Proceedings of the first IEEE Premier Military Communications Conference (MILCOM). pp: 1-7.

[13] *Mininet: An Instant Virtual Network on your Laptop (or other PC).* http://mininet.org

[14] Wang H., Xu L. and Gu G. (2015) *FloodGuard: A DoS Attack Prevention Extension in Software-Defined Networks.* Proceedings of 45th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN). pp: 239-250.

[15] Dhawan M., Poddar R., Mahajan K. and Mann V. (2015) *SPHINX: Detecting Security Attacks in Software-Defined Networks.* Proceedings of the 22th Annual Network and Distributed System Security Symposium (NDSS).

[16] Amann J. and Sommer R. (2015) *Providing Dynamic Control to Passive Network Security Monitoring.* Research in Attacks, Intrusions, and Defenses. pp: 133-152.

[17] Mekky H., Hao F., Mukherjee S., Zhang Z. and Lakshman T. (2014) *Application-aware Data Plane Processing in SDN.* Proceedings of the Third ACM Workshop on Hot Topics in Software Defined Networking. pp: 13-18.

[18] Berman M., Chase J., Landweber L., Nakao A., Ott M., Raychaudhuri D., Ricci R. and Seskar I. (2014) *GENI: A Federated Testbed for Innovative Network Experiments.* Computer Networks. Vol. 63 (Special issue on Future Internet Testbeds-Part I). pp: 5-23. doihttp://dx.doi.org/10.1016/j.bjp.2013.12.037.

[19] Roesch Martin et all. (1999) *SNORT-LIGHTWEIGHT INTRUSION DETECTION FOR NETWORKS.* USENIX LISA. Vol. 99. pp: 229-238.

[20] Vern Paxson. (1999) *BRO: A System for Detecting Network Intruders in Real-Time.* Computer Networks. Vol. 31. pp: 2435-2463. http://www.icir.org/vern/papers/bro-CN99.pdf

[21] Avallone S., Guadagno S., Emma D., Pescap A. and Ventre G. (2004) *D-ITG: Distributed Internet Traffic Generator.* Proceedings of First International Conference on the Quantitative Evaluation of Systems (QEST). pp: 316-317.

[22] Tirumala A., Qin F., Dugan J., Ferguson J. and Gibbs K. (2014) *iPerf: The TCP/UDP Bandwidth Measurement Tool.* http://dast.nlanr.net/Projects,

[23] Hong S., Xu L., Wang H. and GU G. (2015) *Poisoning Network Visibility in Software-Defined Networks: New Attacks and Countermeasures.* Proceedings of the 22th Annual Network and Distributed System Security Symposium (NDSS).

[24] *Floodlight OpenFlow Controller. Project Floodlight.* http://www.projectfloodlight.org/floodlight/

[25] *The CAIDA UCSD DDoS Attack - 2007 Dataset.* https://www.caida.org/data/passive/ddos-20070804_dataset.xml.

[26] Ballard J., Rae I. and Akella A. (2010) *Extensible and Scalable Network Monitoring Using OpenSAFE.* Proceedings of the Internet Network Management Workshop/ Workshop on Research on Enterprise Networking (INM/WREN), in conjunction with the 7th USENIX Symposium on Networked Systems Design and Implementation (NSDI).

[27] Giotis K., Argyropoulos C., Androulidakis G., Kalogeras D. and Maglaris V. (2014) *Combining OpenFlow and sFlow for an effective and scalable anomaly detection and mitigation mechanism on SDN environments.* The International Journal of Computer and Telecommunications Networking. Vol. 62. pp: 122-136.

[28] Curtis A., Kim W. and Yalagandula P. (2011) *Mahout: Low-Overhead Datacenter Traffic Management using End-Host-Based Elephant Detection.* In Proceedings of the IEEE International Conference on Computer Communications (INFOCOM). pp: 1629-1637.

[29] Shin S., Porras P., Yegneswaran V., Fong M., GU G. and Tyson M. (2013) *FRESCO: Modular Composable Security Services for Software-Defined Networks.* Proceedings of Network Distributed Security Symposium (NDSS), San Diego, CA, USA.

[30] Shin S., Yegneswaran V., Porras P. and GU G. (2013) *AVANT-GUARD: Scalable and Vigilant Switch Flow Management in Software-Defined Networks.* Proceedings of the 20th ACM Conference on Computer and Communications Security (CCS). pp: 413-424.

[31] Khurshid A., Zou X., Zhou W., Caesar M. and Godfrey P. B. (2013) *VeriFlow: Verifying Network-Wide Invariants in Real Time.* Presented as part of the 10th USENIX Symposium on Networked Systems Design and Implementation (NSDI). pp: 15-27

[32] Kazemian P., Chang M., Zeng H., Varghese G., Mckeown N. and Whyte S. (2014) *Real Time Network Policy Checking using Header Space Analysis.* Proceedings of the 10th USENIX Symposium on Networked Systems Design and Implementation (NSDI). pp: 99-111.

[33] Porras P., Shin S., Yegneswaran V., Fong M., Tyson M. and GU G. (2014) *A Security Enforcement Kernel for OpenFlow Networks.* Proceedings of the first ACM SIGCOMM Workshop on Hot Topics in Software Defined Networking (ACM HotSDN). pp: 121-126.

[34] Shin S. and Gu G. (2013) *Attacking Software-Defined Networks: A First Feasibility Study.* Proceedings of the second ACM SIGCOMM workshop on Hot topics in software defined networking (ACM HotSDN). pp: 165-166.

[35] Wang R., Jia Z. and Ju L. (2015) *An Entropy-Based Distributed DDoS Detection Mechanism in Software-Defined Networking.* Proceedings of the IEEE Trustcom/BigDataSE/ISPA. Vol. 1. pp: 310-317.

[36] Shin S., Song Y., Lee T., Lee S., Ching J., Porras P., Yegneswaran V., Noah J. and Kang B. B. (2014) *Rosemary: A Robust, Secure, and High-Performance Network Operating System.* Proceedings of the 21th ACM Conference on Computer and Communications Security (CCS). pp: 87-89.

[37] Chung C-J., Khatkar P., Xing T., Lee J. and Huang D. (2013) *NICE: Network intrusion Detection and Countermeasure Selection in Virtual Network Systems.* IEEE Transactions on Dependable and Secure Computing. Vol. 10. pp: 198-211.

[38] Scott-Hayward S., Natarajan S. and Sezer S. (2016) *A Survey of Security in Software Defined Networks.* IEEE Communications Surveys Tutorials. Vol. 18. pp: 623-654. doi=10.1109/COMST.2015.2453114

[39] Alicherry M., Muthuprasanna M. and Kumar V. (2006) *High Speed Pattern Matching for Network IDS/IPS.* Proceedings of the 14th IEEE International Conference on Network Protocols (ICNP'06). pp: 187-196.

[40] Dharmapurikar S. and Lockwood J. W. (2006) *Fast and Scalable Pattern Matching for Network Intrusion Detection Systems.* IEEE Journal on Selected Areas in Communications. Vol. 24. pp: 1781-1792.

[41] Xiong K. (2008) *Resource optimization and security for distributed computing*, https://repository.lib.ncsu.edu/handle/1840.16/3581.

[42] Chin T., Mountrouidou X., Li X. and Xiong K. (2015) *Selective packet inspection to detect DoS flooding using software defined networking SDN.* The 35th IEEE International Conference on Distributed Computing Systems Workshops (ICDCSW). pp: 95-99.

[43] Xiong K. (2009) *Multiple priority customer service guarantees in cluster computing.* IEEE International Symposium on Parallel & Distributed Processing (IPDPS). pp: 1-12.

[44] Boyer R. S and Moore J S. (1977) *A fast string searching algorithm.* Communications of the ACM. Vol. 20 pp: 762-772.

[45] Xiong K. (2014) *Resource optimization and security for cloud services*, Wiley-ISTE.

[46] Sonchack J., Smith J., Aviv A. J and Keller E. (2016) *Enabling Practical Software-defined Networking Security Applications with OFX.* Network and Distributed System Security Symposium (NDSS). Vol. 16. pp: 1-15.

[47] Porras P. A, Cheung S., Fong M. W, Skinner K. and Yegneswaran V. (2015) *Securing the Software Defined Network Control Layer.* Network and Distributed System Security Symposium (NDSS).

[48] Ahmad I., Namal S., Ylianttila M. and Gurtov A. (2015) *Security in software defined networks: A survey.* IEEE Communications Surveys & Tutorials. Vol. 17. pp: 2317-2346.

[49] Xing T., Huang D., Xu L., Chung C. and Khatkar P. (2013) *Snortflow: A openflow-based intrusion prevention system in cloud environment.* Second IEEE Research and Educational Experiment Workshop (GREE). pp: 89-92..

[50] Xing T., Xiong Z., Huang D. and Medhi D. (2014) *SDNIPS: Enabling Software-Defined Networking based intrusion prevention system in clouds.* 10th IEEE International Conference on Network and Service Management (CNSM). pp: 308-311.