Computer Networks 104 (2016) 27–42

Contents lists available at ScienceDirect

Computer Networks

journal homepage: www.elsevier.com/locate/comnet

A transparent and scalable anomaly-based DoS detection method

Ognjen Joldzic[a,*], Zoran Djuric[a], Pavle Vuletic[b,c]

[a] Faculty of Electrical Engineering, University of Banja Luka, Patre 5, 78000 Banja Luka, Bosnia and Herzegovina
[b] University of Belgrade, School of Electrical Engineering, Bulevar Kralja Aleksandra 73, 11000 Belgrade, Serbia
[c] AMRES - Serbian National Research and Education Network, Bulevar Kralja Aleksandra 90, 11000 Belgrade, Serbia

ARTICLE INFO

Article history:
Received 2 August 2015
Revised 13 January 2016
Accepted 3 May 2016
Available online 4 May 2016

MSC:
68M14
68W05

Keywords:
Intrusion detection
Intrusion prevention
Distributed processing
Load balancing
Security

ABSTRACT

Intrusions and intrusive behaviour can be aimed at different parts of the system, ranging from lower-level network attacks intended to disrupt the flow of data in general, to higher-level attacks targeted against specific applications or services. Due to the constant growth of network traffic and the need to inspect the traffic thoroughly, intrusion detection and prevention are becoming increasingly complex and require significant computational resources. This paper presents a distributed, scalable solution for the detection of lower-level Denial-of-Service (DoS) attacks which are executed by transmitting overwhelming amounts of data with the intention of disrupting regular network service. Scalability is achieved by active traffic balancing among multiple traffic processors, exploiting the flexibility and network programmability that Software Defined Networking paradigm brings and packet processing based on device polling. Traffic processors can be elastically added into the pool depending on the traffic volume. The whole system is completely transparent to the external observers. The paper shows that the implemented balancing algorithm further improves the reliability of the intrusion detection.

1. Introduction

With the constant advances in information and network technologies, the entire landscape of computer networks has changed significantly in recent years. Various new types of client devices and available content and offered services are only few of the factors that contributed to the current state of technology in this field. At the same time, these factors have entailed a number of changes in network design and the set of network technologies needed to support the increased complexity of the environment, new needs and a larger user base. Contemporary networks support simultaneous transfer of various content and services multiplexed over a single physical infrastructure. The number of services supported by this converged network infrastructure is constantly growing as is our reliance on them. Therefore, one of the most important aspects of network implementation which is affected by this converged network model is security [1]. Modern security protocols and techniques have to be robust enough to filter anomalous behaviour from a large volume of very diverse traffic. This paper presents a solution for detecting and mitigating Denial-of-Service (DoS) attacks, which are among the most common types of attacks aimed at the disruption of service operations and reliability. DoS attacks are generated by overwhelming amounts of unwanted network traffic or by forcing computing resources to waste processing and/or storage capacities on unwanted tasks. Detecting DoS attacks requires processing large volumes of traffic and its detailed inspection. This can present a bottleneck and disrupt the other services itself if inserted along the traffic path and if the computing power of the detectors is not sufficient. Therefore, creating a scalable solution for attack detection, capable to be deployed in the most challenging environments is required. Recently, a new network management paradigm, Software Defined Networking (SDN), emerged. It allows flexible and adaptable definitions of the packet forwarding and rewriting rules, and network element programmability from a central controller. These properties enable the design of the attack detection systems that can easily split the load to multiple processors and implement feedback actions that can stop the attack. The solution described in this paper leverages the advantages of multicore processing and a proposed load balancing mechanism, which enable it to overcome the limitations of the standard approach to network threat prevention, and to maintain security without disrupting normal network operation.

The rest of the paper is organized as follows. Section 2 discusses the common characteristics and issues of intrusion detection and prevention. Section 3 shows a detailed view of the current state of technology and research in this field. All parts of the proposed solution and the underlying technologies are described

* Corresponding author.
E-mail addresses: ognjen.joldzic@etfbl.net (O. Joldzic), zoran.djuric@etfbl.net (Z. Djuric), pavle.vuletic@etf.bg.ac.rs (P. Vuletic).

http://dx.doi.org/10.1016/j.comnet.2016.05.004
1389-1286/

in Section 4. This section is divided into 6 subsections, each discussing a distinct segment of the proposed solution: System architecture, software components, communication protocol, load balancing mechanisms and, finally, the detection algorithm and attack mitigation. Section 5 contains an overview of the experiments conducted to prove the efficiency of the platform and the results obtained by those experiments. Finally, at the end of the paper some concluding remarks and outlines of the future work are given.

2. Intrusion detection and prevention - characteristics and common issues

Any action performed by one or more hosts on the network aimed at disrupting the normal operation of the network or obtaining otherwise restricted capabilities or information can be considered as an intrusion. Intrusions and intrusive behaviour can be aimed at different parts of the system, from lower-level network attacks intended to disrupt the flow of data in general, to higher-level attacks targeted against specific applications or services. One of the main issues with intrusion detection and prevention is that each type of attack is specific enough to make it almost impossible to create a general-purpose solution that would provide full protection against all types of known and future attacks [2]. One of the main goals of network attacks is to disrupt regular network activity by exploiting protocol vulnerabilities or by sending a large volume of data (i.e. flooding) or specifically created service requests to overwhelm the receiver's network connectivity or processing resources and cause a denial of service to legitimate users. The attack can be initiated by a single sender, or by a number of compromised hosts (bots) coordinated by the attacker. The latter variant is identified as a Distributed Denial of Service (DDoS) attack.

Although similar in execution and their effects, distributed and non-distributed DoS attacks present an important difference from the standpoint of the security device that is supposed to detect and mitigate the attacks. As the number of attack sources is usually very large, the amount of traffic generated by a single source may often be similar in volume and pattern to regular traffic, and thus difficult to detect. It is only when the attack traffic is aggregated at the entry point of the target network that its destructive potential becomes obvious. As a result, there are no widely accepted procedures for preventing DDoS attacks. Some designs have been proposed in literature that would require an Internet-wide deployment of security nodes and proprietary protocols in order to enable efficient detection and mitigation [3]. These protocols would enable an ISP to remotely enforce security policies based on detected attacks, although the compromised hosts may belong to networks which are administered by different ISPs. The mitigation approach involving inter-ISP communication, although currently implemented in a large number of networks [4], may be challenging to implement for all service providers (especially for smaller providers who have no commercial incentive or resources for implementation of such solutions), which arguably leads to reduced efficiency [5].

Intrusion detection systems (IDS) function as passive monitoring devices that alert the network administrator or another network device in case of suspicious network activity. On the other hand, Intrusion Prevention Systems (IPS) actively participates in the flow of traffic, because the process of intrusion prevention requires them to control the transmission of data.

Both types of security devices can be classified by their approach to traffic analysis or by their threat detection methodology. One of the frequently used traffic analysis techniques is the deep packet inspection (DPI), which provides the widest range of possibilities for detection, because the detection process can be based on any information contained in either the header or the payload of the packet. In recent period, another approach to traffic analysis, called the flow-based detection, has seen a significant development and increase in popularity, despite certain inherent performance issues concerning flow export mechanisms and flow sampling in high speed networks [6]. However, several extensions have been proposed that should increase the performance of flow-based attack detection solutions, especially in the field of DDoS attacks [7].

Depending on the threat detection methodology, intrusion detection can be either signature- or anomaly-based [8]. Signature-based detection algorithms assume that every attack can be, with varying precision, described by a set of rules and packet patterns that are compared to every incoming packet by the scanner in real time. Therefore, the attack signature has to be known before the attack happens in order to correctly configure the matching pattern. In reality, this is often not the case and such strategy can be easily exploited by deducing which patterns are recognized by the detection device, and modifying and executing the attack accordingly.

The other major group of detection algorithms, anomaly-based, relies on the fact that, during the attack, one or more network parameters values will significantly differ from the measured baseline. These algorithms require a scanner to undergo a learning period in order to establish the baseline values. Anomaly-based detection requires less configuration by the administrator and is arguably more robust in terms of discovering attacks that present themselves in previously unknown attack patterns [9].

Regardless of the attack detection approach, the IPS must perform its function without introducing any significant latency, since the service levels of the protected network (the network behind the security appliance) are directly affected by the inspection. IDS devices have a slightly higher tolerance for latency, since they usually receive a copy of the traffic which is forwarded along the original path.

The most common point of deployment for an IPS/IDS within the protected network (i.e. the network which is the attack target) is as close as possible to the probable source of the attack. This deployment increases the chance for a positive detection, since the IPS/IDS device has access to the aggregated traffic that enables it to obtain a complete view of the attack in progress [5]. On the other hand, the placement of the device makes it a possible bottleneck and a single point of failure for high speed networks. Therefore, a logical solution to this issue is the introduction of a scalable device that would distribute the processor load required for packet processing to a number of processors, while still maintaining effectiveness.

The solution proposed in this paper is a transparent intrusion detection system (TIDS), developed using the anomaly-based attack detection methodology. The design goals, operation and structure of the proposed system is explained in detail in Section 4.

3. Background and related work

Contemporary security appliances (built either for intrusion detection or prevention) must operate in near real time in order to be able to efficiently respond to threats involving large volumes of traffic commonly found in today's networks. This section highlights the recent work in the domains of parallel traffic processing and scaling to large volumes of traffic during peak usage periods.

Lakhina et al. [10] discuss the possibilities of detecting various attacks based on the entropy values of certain packet header fields of the incoming traffic. The same approach has been taken by Giotis et al. [11], but with the introduction of SDN to increase the programmability and flexibility of the solution. Extending these findings, the solution proposed in this paper is a highly scalable multi-processor detection system based primarily on entropy calculations.

If the detection process is based on packet inspection, the operation of IPS and IDS consumes significant processing power and adds to the latency of the traffic if inserted on the traffic path. Therefore, providing efficient packet processing is a crucial prerequisite for such IDS and IPS systems. This can be achieved by spreading the load to multiple network traffic processors and by efficient packet processing within the processors. The majority of load balancing algorithms are tailored to work with network environments that have specific host configurations (e.g. data centres with distributed web servers etc.). For example, several approaches deal with the distribution of client requests across a number of equal servers in the datastore [12,13]. The proposed algorithms often require the balancer to alter the addressing information within the packets thereby rerouting the traffic to the appropriate destination. Lai et al. [14] presented a solution which employs a hardware-based IDS containing a special scattering component. This component distributes the traffic across a number of sensors which are responsible for packet inspection and traffic forwarding. In order to increase the efficiency of a system which employs load balancing, the communication between the receiving nodes must be reduced to a minimum, which is a goal often underlined in research [15], [16].

The solution described in this paper relies heavily on the benefits provided by Software Defined Networking, which provides high flexibility and usability in various applications [17–20]. Its configurability and programmability make it a suitable platform for implementing adaptive network nodes that can easily respond to changes in network environment and topology. Research published in [21] and [22] show the use of policy enforcement and load balancing in SDN networks, respectively. However, the performance of SDN is one of the biggest shortcomings which is often underlined in literature [23,24], especially for high speed networks that employ reactive flow processing or a large flow granularity. TIDS solves this problem by delegating the most of the complex calculations to a number of processors, thereby removing the load from the SDN controller.

4. Design goals, operation and structure of TIDS

4.1. Design goals

The main design goal of the solution presented in this paper (TIDS - Transparent Intrusion Detection System) is to provide a proactive, scalable and secure system built on commodity hardware. The system primarily works as an IDS, but also provides the attack prevention functionality for non-distributed attacks. The security aspect of the system is achieved through its transparency (invisibility) to other nodes within the network by maintaining full control over the traffic that passes through the system, thereby making it resistant to attacks aimed against the system itself.

The scalability aspect is achieved through distribution of its processing logic to an arbitrary number of traffic processors using a load balancing algorithm. The traffic processors, coupled with SDN components, use an anomaly-based attack detection methodology to detect (D)DoS attacks in high speed network environments. It is absolutely necessary that the solution performs without disruptions to the normal operation of the network, regardless of the bandwidth or any other characteristics.

4.2. System architecture

TIDS consists of two SDN network nodes (switches), the main SDN controller and a variable number of packet processors (scanners), as displayed in Fig. 1.

SDN switches (S_1 and S_2 in Fig. 1) provide the entry and exit point into the IDS, respectively. S_1 is responsible for distributing and policing traffic as it enters TIDS from the outside network (on the left side in Fig. 1), whereas S_2 aggregates the traffic from a variable number of processing nodes and forwards it into the internal (protected) network.

TIDS is designed as a detector for intrusion attacks originating from the external network. As the traffic enters TIDS, it is spread across the active processing nodes (processors or scanners, shown in Fig. 1 as P_1, P_2, etc.). The algorithm used in TIDS requires only the data contained within the addressing fields of Layer 3 PDUs, so the system doesn't necessarily have to be packet-based, nor does it need to perform full Deep Packet Inspection in order to function. Each node processes the incoming traffic, and determines whether an attack is in progress by using an entropy-based algorithm, described in Section 4.5. At the same time, should any of the processors experience a higher than normal traffic load, the traffic distribution patterns on switch S_1 are reconfigured to equalize the amount of traffic directed to each of the processors. This facilitates the detection process since the TIDS uses SDN only to perform the policing and the distribution of traffic, but the computationally intensive operations (such as packet inspection and calculations) are delegated to the processors.

The traffic that flows through TIDS can be divided into three groups: management, control and regular traffic. Management traffic carries management messages between the processing nodes and the controller C_1, and is present only between C_1 and the processors. Regular traffic is the traffic that is subject to analysis, and is always originated by hosts outside of TIDS. All links between S_1 and S_2, with the exception of the direct link (shown at the bottom of Fig. 1), carry the regular traffic flows from S_1 towards S_2, passing through one of the processing nodes along the path. On the other hand, the direct connection is used to forward traffic that is not being analysed, but has to exist on the link between the two devices (this traffic forms the control traffic group). Aside from control traffic, which includes different kinds of locally generated broadcast and unicast traffic, this link is used to forward outbound traffic from the internal network (newly originated requests or responses to earlier requests). It is assumed that the traffic originating from the protected network is safe and that no processing for this traffic is required. In this sense, switches implement asymmetric routing - the return traffic from the internal network can be forwarded outwards without delays. This simplifies the forwarding procedures, as the load balancing logic is only required on S_1, whereas S_2 only aggregates and forwards the traffic. In the cases where the protected network could contain an attack source, another instance of TIDS might be placed on the same link facing the opposite direction (i.e. processing the traffic that originates from the internal network). Presumably, the smaller number of hosts within the protected network (compared to the number of hosts on the Internet) would simplify the detection of malicious activity, since the change in entropy would be more prominent.

The direct link between S_1 and S_2 switches exists as a fallback connection regardless of the number of active processing nodes. This ensures that in the case of processing node malfunction, no traffic is dropped due to the lack of forwarding path.

The controller C_1 is the only node communicating directly with the processors. It monitors the active nodes, and reconfigures the load distribution dynamically as the number of active processors changes during the operation. There is no communication between the processing nodes, which means that the traffic has to be partitioned in a way that each node has enough information to make a positive identification for an attempted (D)DoS attack. The processors have no knowledge of the portion of the incoming traffic it will receive, and do not actively participate in the redistribution of traffic.

The traffic processors are positioned as pass-through devices that inspect and forward traffic along its original path. The

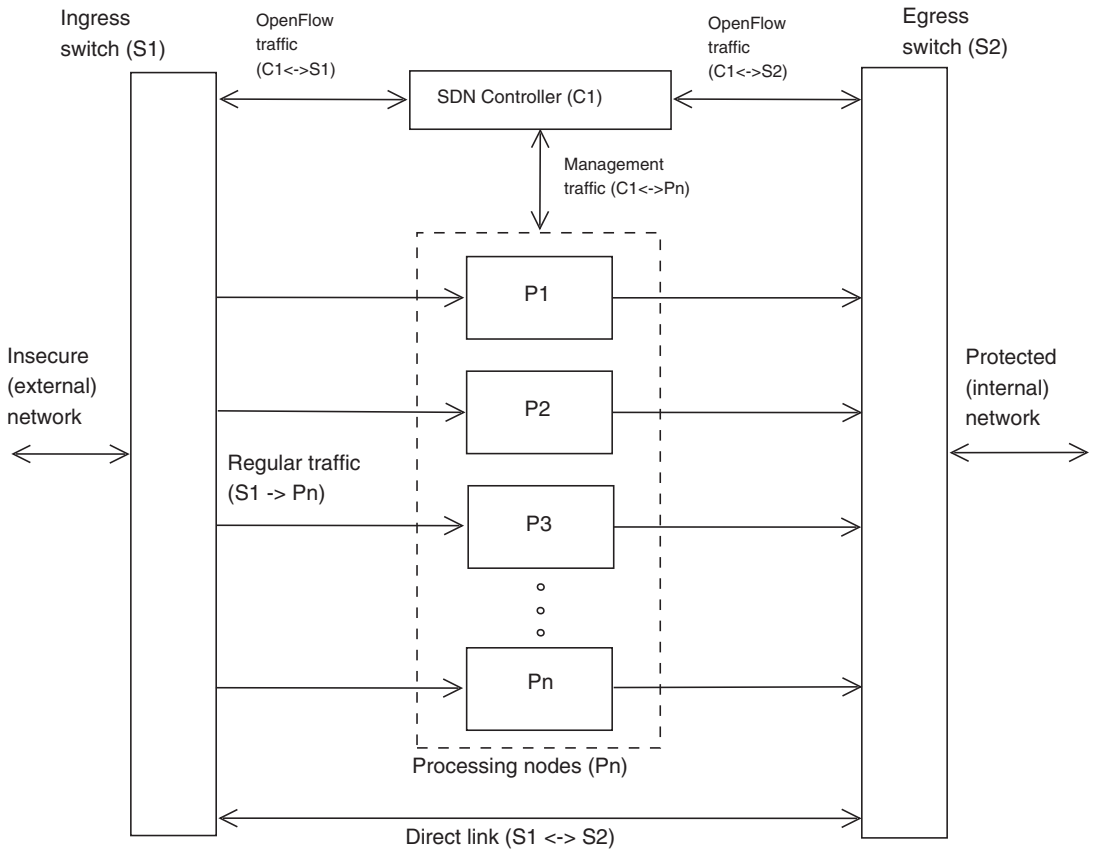

Fig. 1. The main architecture of TIDS.

information obtained by inspecting the packets is used to calculate entropy values for destination addresses and source- and destination-based IP prefix distributions. After the processor has made a positive detection, the set of addresses from which the suspected violation is coming is communicated to the controller, which instates one or more flow modifications on S_1 in an attempt to mitigate the attack.

4.3. Software components

There are two major software components used in TIDS. The first component is the SDN controller software that is used to implement the control plane of a network device. TIDS uses the Ryu SDN controller [25].

The processors are the main packet capturing and processing components of the entire system. Packet capturing is typically based on interrupts generated by the network interface card (NIC), which are serviced by a routine that performs the capture. For lower speed networks, this mode of operation is sufficient enough to provide a capturing framework that has an insignificant packet loss rate, and that no major performance gain is achieved by utilizing a different capturing software layer [26]. However, in high speed environments, the amount of time spent servicing the interrupt request is far too expensive for any application.

Device polling is a technology that can mitigate partially or completely the slow response of the interrupt based packet processing in environments where a low latency (or high throughput) is required. Instead of spending long periods of time servicing interrupt requests originated by the NIC, the processor masks all further interrupts generated by the card, and allocates a task that polls the NIC in regular intervals. This results in a significant improvement in efficiency and speed of packet processing [26]. An example of a device polling library is Intel's DPDK library [27], which is used for the implementation of processors. DPDK is a set of user-space libraries that function above an Environment Abstraction Layer (EAL) in order to provide a common development platform for poll-mode network applications. Performance reports and Intel's white sheets show that DPDK can withstand high network loads of more than 40 Gbps [28]. DPDK does not impose any special requirements for implementation other than a compatible NIC and a Linux-based operating system. This enables the processors to be implemented as virtual devices (as is the case with the testing environment created for the purpose of this paper). In virtual environments, the capacity of the system (in terms of available bandwidth) is limited by the capabilities of the virtualization environment, and not TIDS.

As a consequence of the programmability of the system and the control it maintains over the contents of different packet header fields, the entire system can be completely transparent to other network devices. Frames are forwarded without modification to any of the fields inside the network and data link layer PDU, thus hiding the fact that the packet has traversed one or more nodes on its path. This also includes the control traffic between two adjacent network nodes that have physically been separated by TIDS. Therefore, nodes within the TIDS system require no addressing information (aside from a possible management interface which is not considered part of the system). This avoids an entire class of network threats that target the IPS system itself, attempting to disconnect the entire network behind the IPS from the rest of the world [29].

4.4. Communication protocol

The communication protocol used between the processors and the controller is a lightweight protocol which carries management data. Management messages are used by the SDN controller to request additional information from the processors, or by the processors to request the controller to modify the flow table on S_1 by sending Flow Modification messages. As mentioned previously, messages that are sent between the processors and the controller are confined to the portion of the network between the processors and the controller. The messages are encapsulated in an Ethernet header with the ether-type field set to the currently unused value of 0x9009 [30], which allows a clear distinction of the protocol from any existing Layer 3 protocol. Each message contains a type field, indicating the encapsulated command. The following message types are defined:

- Keepalive message (type 1) - used to announce the presence of the processor to the controller. The processors send keepalive messages in regular short intervals of configurable length. Upon the reception of the first keepalive message from the processor, it adds the newly discovered processor to its local database, and adds the corresponding port to a list of active ports. This enables the administrator to expand the system's capabilities by simply adding a required number of processors and connecting them to S_1. If any of the processors fails to send 3 consecutive keepalive messages, it is declared unreachable by the controller and the traffic flow is reconfigured to reflect that change. The keepalive messages also carry a packet count for the previous interval, which is used for load balancing purposes (the effect of the length of the interval is discussed in detail in Section 4.7),
- Shutdown message (type 2) - used to alert the controller that the processor is shutting down and will no longer be active on that particular port. The port is removed from the list of active ports, and the flows are recalculated accordingly,
- Flow modification request (type 3) - used to trigger a modification of the flow database. These messages are sent by the processor that discovers an attack in progress, and contain one or more network addresses that are involved in the attack. The controller, in turn, creates the required blocking rules based on the contents of the message. A detailed description of the attack detection process is given in Section 4.5,
- Prefix list request/response message (type 4) - used by both the controller and the processors to transfer detailed information about network prefixes and packet counts. By sending this message, the controller requests a detailed list of destination networks from the processor, to which at least one packet had been sent during the previous measurement interval. In response, the processor creates a list containing packet counts for each of the prefixes along with the network address and forwards it to the controller. The list is used by the controller to recalculate the network load, as described in detail in Section 4.4. Although the SDN controller can obtain the statistics for each of the processors from the switch, this approach reduces the load on the switch and enables the processors to maintain separate packet counts for different network prefixes. This information is not available on the switch in the early stages of the load balancing, due to the larger flow granularity than required by the processors. Furthermore, research shows that SDN components may display certain imprecisions in accounting [31] which may have an undesired effect on the validity of the results.

4.5. Detection algorithm

The detection algorithm is based on the entropy value calculated for the destination addresses of the traffic each processor receives from the switch. The entropy is calculated using the Shannon entropy calculation method, as shown in (1), where p denotes the probability of the occurrence of an IP address as the destination of a packet.

$$H = -\sum p * \log_2 p \tag{1}$$

The main idea behind the algorithm is that the decrease in the entropy value for the destination addresses shows that the amount of traffic directed towards a small number of hosts has increased and that it now occupies a significant portion of total traffic. If the decrease in entropy is sharp (it occurs over a short interval) this may signal that a (D)DoS attack has begun. Although the entropy-based intrusion detection is a familiar concept in literature, the novelty of the approach presented in this paper relies in the distribution of the processing logic to an arbitrary number of nodes. Namely, the difference in entropy is more apparent when the attack traffic takes up a larger portion of the total amount of traffic (or when the intensity of the background traffic is smaller compared to the attack traffic). By distributing the total amount of traffic to a number of processing nodes, TIDS reduces the amount of background traffic that is processed by each processor, thereby emphasizing the anomalous traffic and simplifying its detection. Additionally, the balancing procedure is temporarily suspended during suspicious network activity (i.e. when one of the processors signals that a possible attack is currently in progress), in order to prevent the redistribution of the attack traffic among the processors. The redistribution would decrease the attack traffic intensity on the processor which detected the suspicious activity, thereby increasing the entropy value, which would reduce the chances for a successful attack detection.

Similarly to the other entropy-based anomaly detection systems [10,11], TIDS divides the monitoring interval into time windows (also known as slices), which contain packet statistics for that time interval. The duration of individual slices affects the robustness and responsiveness of the system. Longer time slices provide resistance to false positive detections that may appear due to short spikes in network traffic, whereas shorter time slices increase the responsiveness of the system. The duration of the slice of 30 s gave satisfactory results for experiments given in Section 5.3.

These slices are used by the processors to store the entropy values and the packet counts for each destination prefix. The slices are placed in a circular list, so that no additional memory has to be reserved as the slices are filled with data. After the expiration of the last interval, the oldest time slice is cleared and used for new data. The total number of slices is larger than the maximum number of slices needed for a positive detection (the number of slices is given in Section 5) so that the required data is not deleted while it is still needed by the processor.

Each processor works as a finite state machine (as shown in Fig. 2), with three distinct modes of operation:

- Idle mode of operation is a state which is entered upon system startup or when the processor receives no data for a short period of time. The duration of this mode of operation affects the detection procedure, and should be a small fraction of the length of the time slice. While in this state, the processor does not perform entropy calculations, since the baseline would be incorrectly interpreted and any traffic that arrives after this state could be considered attack traffic if compared to entropy values when the system is idle,
- Learning mode is a state during which the processor collects data and establishes a baseline. The system always transitions

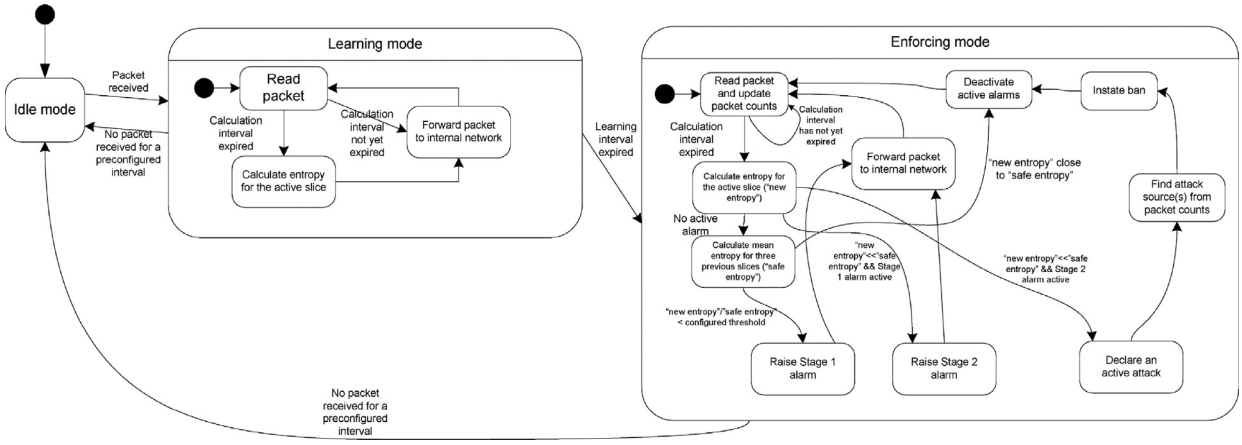

Fig. 2. The detection algorithm state machine.

to this state from idle mode after the receipt of the first packet, and it lasts for a configurable number of time slices. A longer learning mode would take into account a larger number of previous slices when calculating the baseline entropy. Short fluctuations of the entropy value during the learning period would not have a significant effect on the average baseline (because of the larger number of slices in the learning period). This would desensitize the system to shorter bursts of traffic, which may lead to false positive detections. On the other hand, by having a shorter learning period to establish the baseline, the system would be immune to short bursts in traffic volume that do not constitute an attack. The selection of the value needs to be made with regard to the non-attack traffic patterns of the protected network, and it should be long enough to avoid the possibility of accidentally selecting a short burst of traffic as the baseline value upon which the detection algorithm will be based. The length of the learning mode for the purpose of the tests in this paper is set to 3, based on the characteristics of the dataset that is used as the background traffic, which has a rapidly changing traffic pattern. The dataset is described in detail in Section 5.2,

- Enforcing mode is a state during which the processor reacts to any suspicious network activity and detects incoming attacks by comparing current entropy values to the established baseline.

After the processor enters the enforcing mode, it is assumed that a sufficient number of learning intervals have already passed, and that the processor has relevant baseline data. Packet counts for different prefixes for each time slice are kept in a sorted hash map for better search performance. The process of locating the prefix (i.e. the key in the hash map) and storing the packet counts is the most time-consuming calculation that may introduce significant latency and must not be computationally expensive, even in deployments consisting of small number of prefixes. After the slice expires, packet counts from the map are used to calculate the current entropy value. This value is then compared to the mean value from three previous non-attack time slices (also known as the "safe" value), by calculating their ratio. The primary purpose of the initial learning mode is to ensure that the system goes through at least three non-attack time slices. Since there is still insufficient data at that time, the administrator must verify that an attack is not currently in progress during the learning period.

If the calculated entropy ratio is lower than the value of the configurable threshold value (also called the Configured Entropy Ratio (CER)), the processor raises a Stage 1 alarm denoting a possible attack. It is important to note that the CER value is expressed as a percentage of the safe value (the mean value from three previous non-attack slices), thereby providing the system with robustness by enabling it to respond to attacks in an adaptive manner. In the following time slice, the same calculation is repeated (again by comparing the current destination address entropy to the safe mean value). This is done to avoid classifying short bursts of traffic from a single IP address as attacks. If the entropy value persists during this second time slice, then it is legitimately regarded as an attack in progress and the processor raises a Stage 2 alarm. A slight variation between entropy values in alarm stages 1 and 2 is allowed, to prevent minor fluctuations in either background or attack traffic from disrupting the detection process (i.e. a short decrease in volume of the attack traffic that would affect the entropy value). If the new entropy value returns above the configured threshold, the processor will assume the attack has already passed and return the system to a safe state. The number of slices to confirm an attack (2 slices from the first entropy deviation) is chosen to accommodate sharp changes in traffic patterns (that happen within a minute). An additional mechanism is implemented which prevents the attacker from bypassing the detection mechanism by gradually increasing the attack intensity over a longer time period (10 or more time slices). If the changes in entropy and packet count are monotonic during this period, and the final entropy is below the CER threshold, an alarm will be triggered and the system will proceed to the attack mitigation stage.

4.6. Attack mitigation

After an attack has been confirmed, the processor spends another time slice to determine the type of attack. Determining the type of the attack involves counting the packets aimed at the attacked hosts and grouping the counts according to their source IP address. TIDS starts monitoring the source addresses only after an attack has been confirmed and the attacked destination has been identified. This reduces the memory requirements for TIDS, especially during normal operation (without an active attack). A source host is considered suspicious (a possible attacker) by comparing its packet count to the mean packet count for all tracked sources.

If the attack does come from a small number of hosts (i.e. the attack is not distributed), TIDS is able to easily mitigate this attack by blocking the suspicious sources without disrupting the service for regular users. The number of attackers required for an attack to be classified as distributed is configurable, and was set to 50 for the purpose of the experiments in this paper. If the configured value is smaller than the actual number of attack sources whose packet count exceeds the calculated mean count, the system will require more time to fully mitigate the attack (after the first set of attack sources is blocked, TIDS will detect the same attack again,

only this time originating from a different set of attackers). Therefore, there is no upper bound for the value of this parameter, except for the one imposed by the amount of memory available to TIDS.

TIDS is able to simultaneously track a larger number of attacked hosts, and to mitigate all attacks that are currently in progress. After the source and destination of the attack have been isolated, the processor sends a Flow Modification Request message (type 3) containing the list of addresses to be banned. The bans are instated for a limited amount of time, after which they are automatically aged out by the switch. The system employs a penalization scheme which doubles the previous ban length for repeated attacks that are originated from the same host (similar in implementation to the BGP route dampening penalization scheme [32]), in order to minimize the effects of repeated attacks. The initial ban length is also configurable (set to 300 s for the purpose of the tests), and should at least be longer than the time required to detect an attack.

In case of distributed DoS attacks, the attack typically comes from a large number of source addresses. It is virtually impossible to completely eradicate this type of attack by blocking the attacking hosts from reaching the target, since it is difficult to isolate the attackers from the legitimate users without analysing the application-layer data [33]. However, as previously stated, this would significantly increase the complexity of the system and degrade its performance. The inability to differentiate malicious from regular users is caused by the fact that a single attacking host performs only a handful or requests - usually not more than an ordinary user that is not part of the attack. Any attempt to block the attack traffic from reaching the target by using only network layer information would either have to employ one of the Internet-wide security solutions already discussed in Section 3 [3,5], or be based on the destination address (which would mean disconnecting the valid hosts from the service as well as the attackers). The latter approach is also known in literature as blackholing [34]. The issue of false positive detections and DDoS attacker isolation is further discussed in Section 5.4.3.

Another important point that must be addressed in this section is the process of recovery after an attack is detected and proper action has been taken in response. With the load balancing algorithm temporarily suspended after the sign of an attack, enough time can pass until it is restored again so that the distribution rules can be considered stale. In order to speed up the recovery process, the controller removes all additional rules and starts the distribution procedure from the beginning (i.e. the current distribution rules are flushed, but the previously instated bans remain in place). This has no effect on the detection process, but speeds up the process of converging to a balanced state.

4.7. Load balancing algorithm

Load balancing is a technique for capacity scaling often used in computer networks either as a part of the link aggregation groups or when routing protocols calculate the same metric over more than one path simultaneously. As load balancers are placed on the path of the traffic, load balancing algorithms have to be efficient and to add negligible latency. Therefore, stateless hash based techniques which as an input into the hash function take some key parts of the packet header, and distribute the packet flows to several outputs are used most often [35–37]. The efficiency of the algorithm depends on the statistical properties of the traffic and the matching algorithm. The majority of the flows comprising network traffic are short lived flows which do not contribute significantly individually to the traffic volume (more than 80% of the flows last less than 10 s) [38]. However, there is a small percentage of long lived flows that consume significant percentage of the link capacity. The existence of those long lived flows present significant challenge to the task of distributing the traffic evenly over several paths, as any hash based matching algorithm can hardly split such flow over parallel paths or to match its throughput with other flows being transferred over the alternative paths. RFC 7424 [39] explores some guidelines to achieve as close to optimal as possible load balancing.

The algorithm presented in this paper was developed to equalize the load on each of the active processors and to enable the administrator to increase the capacity of the system by including additional processors. At the same time, it is designed not to interfere with the intrusion detection procedures or to put unnecessary strain on the SDN controller itself. The system assumes that any attempted (D)DoS attack is aimed at one or more particular hosts inside the protected network. Therefore, any traffic destined to the same host will be forwarded over the same component link and processed by the same processor at any single time point. It can only be rebalanced to a different processor if the system concludes that there is no risk of an attack. Based on its operation, this type of load balancing is usually classified as server-based, as opposed to the per-request, per-flow or per-destination solutions found in literature [12,40–42].

In high speed environments, processing each new flow individually and reactive installation of flow rules triggered by the incoming traffic may prove to be computationally expensive [43]. Hence, on controller startup, TIDS proactively generates fallback forwarding rules that partition the traffic on regions based on the low order bits of the destination IP address, with the number of bits used in the mask depending on the number of active processors (the lowest number of bits n for which 2^n is larger or equal than the number of processors). Although the resulting loads on some processors may be unequal, the number of rules required to adapt the distribution pattern is significantly lower than it would have been if each of the flows had been processed individually. Each new processor (identified by a keepalive message received through a previously inactive port) will cause the controller to rebalance the traffic. These rules have a lower priority, functioning as fallback routes in cases where no specific rule exists.

After the start of operation of TIDS and the establishment of the initial rules, the controller is only responsible for rebalancing one or more prefixes in order to accommodate for any disparities between the loads of different processors. The traffic is rebalanced by rerouting prefixes of configured length (as determined by the Prefix Length (PFS) parameter, explained in Section 5) from one component link to another, based on the current load of those links. Each of the processors keeps track of the number of packets for each different destination network prefix during the observation interval between two consecutive keepalive messages. Every keepalive message sent out by the processor contains a summary count of all packets processed during the previous interval. Upon receipt of a keepalive message from each of the processors, the controller compares the reported packet counts. If a disparity which exceeds the imbalancing threshold is encountered, the controller sends out a prefix list request message (type 4) to all processors for a detailed enumeration of all prefixes and their packet counts. Based on these responses, the controller performs a rebalancing by installing a number of higher priority rules that reroute specific prefixes from a congested component link to one that is currently underutilized. The algorithm selects the prefixes by estimating the packet count after the rerouting procedure is complete (based on the packet count for each prefix for the previous time slice). The prefixes are placed in the list in descending order, based on the packet count, in order to minimize the number of iterations required to rebalance the load. The pseudo-code for the algorithm is given in Algorithm 1.

Algorithm 1 The load balancing algorithm

Require: *actcnt* = Number of active processors, as determined by Keepalive messages, *thresh* = Configured imbalancing threshold, *procs* = List of active processors

numreps ← 0 ▷ Number of received reports
meancnt ← 0 ▷ Mean packet count
for all $p \in procs$ **do** $p.packetcnt \leftarrow 0$ ▷ Packet count for processorInitialize list $p.prefixes$ ▷ Network prefixes and counts per prefix
while $numreps \neq actcnt$ **do**
    update $p.packetcnt$ for the processor p_i
    $numreps \leftarrow numreps + 1$
$meancnt \leftarrow calculatedmeanpacketcount$
Initialize list of *underutilized* processors
Initialize list of *overutilized* processors
for all $p \in procs$ **do**
    if $p.packetcnt > meancnt^*(1+thresh)$ **then**
        add p to *overutilized*
    else if $p.packetcnt < meancnt^*(1-thresh)$ **then**
        add p to *underutilized*
if *underutilized* is not empty and *overutilized* is not empty **then**
    $numreps \leftarrow 0$
    for all $p \in procs$ **do**
        Request detailed prefix list from p
    while $numreps \neq actcnt$ **do**
        update $p.prefixes$ with prefixes and counts per prefix for report from processor p_i
        $numreps \leftarrow numreps + 1$
    for all $o \in overutilized$ **do**
        for all $u \in underutilized$ **do**
            for all $prefix \in o.prefixes$ **do**
                if $o.packetcnt - prefix.packetcnt > meancnt^*(1-thresh)$ and $u.packetcnt + prefix.packetcnt < meancnt^*(1+thresh)$ **then**
                    Remove $prefix$ from $o.prefixes$
                    Add $prefix$ to $u.prefixes$
                    $o.packetcnt = o.packetcnt - prefix.packetcnt$
                    $u.packetcnt = u.packetcnt + prefix.packetcnt$
                    Create Flow modification rule

After the new rules are installed, the controller goes into a rebalancing interval, during which no load recalculations are allowed, regardless of the summary counts. This interval is used to increase the stability of the system by preventing frequent recalculations and forwarding changes. This interval is configurable in TIDS through the value of a parameter called the rebalancing timer (RIT). The effect that the value of RIT has on the system is explained in Section 5.3.1.

5. Experimental results and discussion

Experimental evaluation shows that the introduction of a distributed environment into an entropy-based IDS solution is beneficial in two areas: it reduces the load on the processing nodes and simplifies the attack detection procedure. The load balancing algorithm enables the proper operation of the system regardless of the number of processors or the amount of traffic that passes through the IDS.

5.1. Test network description

The testing environment set up for the purpose of the experiment uses a virtual infrastructure based entirely on commodity hardware. The virtualization technology provides a suitable platform to prove the expandability and scalability of the solution to an arbitrary number of nodes.

The diagram of the actual platform is given in Fig. 3. Each of the processing nodes is based on a single-core Intel Xeon E5-2420 processor, while the switch nodes use the quad-core variant of the same processor. The amounts of available memory on the processing nodes vary between 1 GB and 2 GB, while the switch nodes are equipped with 8 GB of RAM. The operating systems are all Linux-based. The virtualization infrastructure was provided by VMWare ESXi, including the hardware platform and the network connections [44] which are used by the nodes for internal communication and administration. The virtual network uses minimal configuration and functions in promiscuous mode, thus simulating a real-world network environment in which the receiver has virtually no control over the traffic it receives over public interfaces. Network is implemented using Mininet with OpenVSwitch virtual switches which connect the virtual interfaces of the host server. Aside from the load balancing and attack detection procedures, the SDN controller implementation contains a simple MAC learning functionality in order to reduce the amount of unnecessary flooding of traffic. Although the source and target servers were separated by three nodes along each path, simple ICMP messages showed that the servers considered each other to be only a single hop away, which confirmed the transparency premise set up at the beginning of the paper.

5.2. Data sets

The testing of the load balancing and attack detection algorithms was performed using the dataset obtained from the Simpleweb/University of Twente Traffic Data Repository [45], and contains the data collected from a residential network of a university. The dataset contains a mixture of different IP-based application-layer and transport-layer protocols and uses more than 10,000 different IP addresses. The collected packets have been treated with prefix-preserving anonymization prior to publication. The actual packet content is not significant for the test in question. The replay speed of the actual dataset was multiplied to achieve a throughput of around 80 Mbps. This was done by using the tcpreplay utility [46].

Some of the initial testing was also performed using the CAIDA data set provided by the PREDICT repository [47]. However, the average throughput of this data set is significantly lower than the other sets, so the test results, although correct, are not included in this paper. In order for it to be comparable, the packets would have to be replayed at a throughput which is more than 100 times faster than the original, which would create unnatural traffic patterns (e.g. much sharper increases in traffic volume, that would normally occur over a much longer period) that might even be flagged as malicious.

5.3. Experimental results

The tests consisted of generating two DoS attack streams over the background traffic. The attacks were generated by transmitting UDP datagrams at a rate of approximately 15 Mbps (configured on the generator as 40,000 datagrams per second). In order to achieve this, a tool called BoNeSi (DDoS Botnet Simulator) [48] was used, which is able to generate vast amounts of packets from a single host address, or from a set of addresses provided within the configuration files. The latter setup enables the simulation of a distributed attack. The two attacks are started 30 s apart, 10 min after the beginning of the test.

The test results consist of two parts. The first part shows the load of each processor expressed through the number of packets

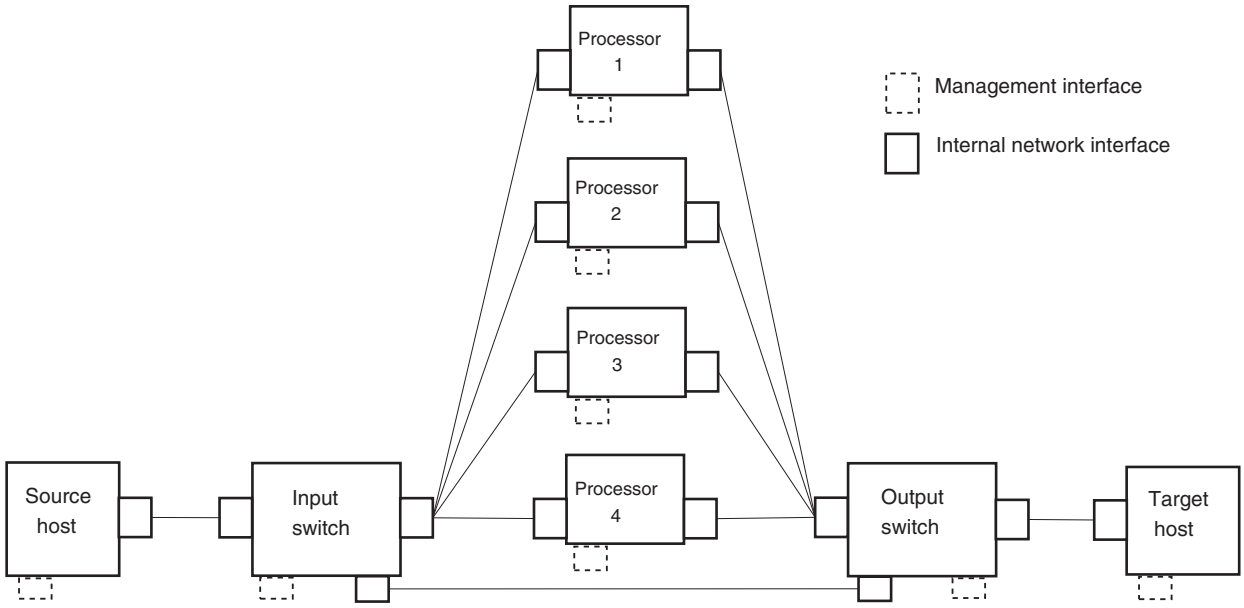

Fig. 3. The testing environment.

processed during the measurement interval, whereas the second part displays the total entropy of the destination prefixes.

Figs. 4a–d show the load of the system in configurations with 1, 2, 3 and 4 processors, respectively. The start of the first attack is visible around the 10th min, whereas the second attack appears 30 s later. After the detection has been made, the traffic volume returns to normal for a period of 10 min. After the ban has expired, the increase in volume is once again visible around the 21:00 and 21:30, respectively. Therefore, for the duration of the test, the system has to detect each of the attacks twice, over the span of ten minutes.

In configurations with more than one processor, the destination address for each of the attacks is configured so that the attack traffic is processed by different processors. Although it cannot be guaranteed that the two attacks will be processed by two particular processors, the tests have shown that the system is able to identify the attack traffic even with the load distribution procedure in action. After the attack is suspected, the load balancing is temporarily suspended and the attack traffic remains on the same processor until mitigation. Otherwise, the load balancing algorithm would attempt to equalize the load on the processors which might affect the entropy values and impede the detection of the attack.

Figs. 5a–d show the entropy fluctuations between and during the attacks. Once again, the figures are aligned with Fig. 4.

Figs. 6a and b show the entropy and load values for a distributed DoS attack using a pool of 50,000 source addresses. The figures show that the distributed nature of the attack has no effect on the detection procedure. However, since it was not possible to distinguish the addresses of the attackers from the legitimate users, the only available action for the mitigation of the attack was to temporarily block all access to the attacked host.

Figs. 7a and b prove the necessity of the load balancing mechanism by showing the load and the entropy in case when the traffic is distributed based only on the lowest bits of the destination address (i.e. no load balancing mechanism is active).

The intensity of the attack for the tests is chosen in order to prove the advantages of the distributed system design. Namely, in a single-processor setup, the attack traffic blends more into the background traffic, thereby creating a gradual change in entropy that is harder to detect than in the cases with a more aggressive attack (i.e. the slope of the entropy graph reaches its lowest value much slower, as shown in Fig. 5). The minimal intensity of the attack which can be detected depends heavily on the parameters of the environment (the number of processors, the characteristics of the background datasets, the available throughput etc.), so the sensitivity of the algorithm cannot be discussed in general. In this particular case, the attacks for the single-processor environment could be detected by adjusting the entropy threshold to a higher value (over 80 % of the “safe” value), thereby increasing the sensitivity of the system. On the other hand, this would increase the risk of classifying a change in entropy that comes from normal network operation (a sudden increase in traffic towards a single host, caused by interesting content, also known as a “flash crowd”) as an attack .

5.3.1. Setting up system parameters

Several additional tests have been conducted in order to provide a validation for the chosen values of configuration parameters for different parts of TIDS.

Figs. 8a–c show the resulting load on the processors for different values of PFS (12, 20 and 28). Smaller values of PFS affect the balancing negatively. Since the network groups are very large, any attempt to equalize the load creates a new inequality, and the system is never able to reach balance (as seen in Fig. 8a). On the other hand, a value that is too small creates a large number of networks that degrade the performance of the system by slowing down the balancing process.

Figs. 9a and b show the behaviour of the system for very long and very short RIT intervals. The RIT parameter affects mainly the stability of the system. In the case of the long RIT values (50 s for Fig. 9a), the difference in processor loads is more evident as more time between two consecutive recalculations. This lowers the responsiveness of the system to quicker changes in traffic patterns. Extremely short intervals, on the other hand, put an additional strain on the controller, since the recalculations and flow modifications have to be executed more frequently. For slow-changing datasets, there is no need to perform the rebalancing too often, as the load on the processors may not be different since the last recalculation (e.g. the RIT value of 15 s yielded satisfactory values for the datasets used in this paper).

The other important parameter for the load-balanced detection mechanism is the prefix size (PFS). The prefix size determines the minimal size of the network prefix that can be rerouted

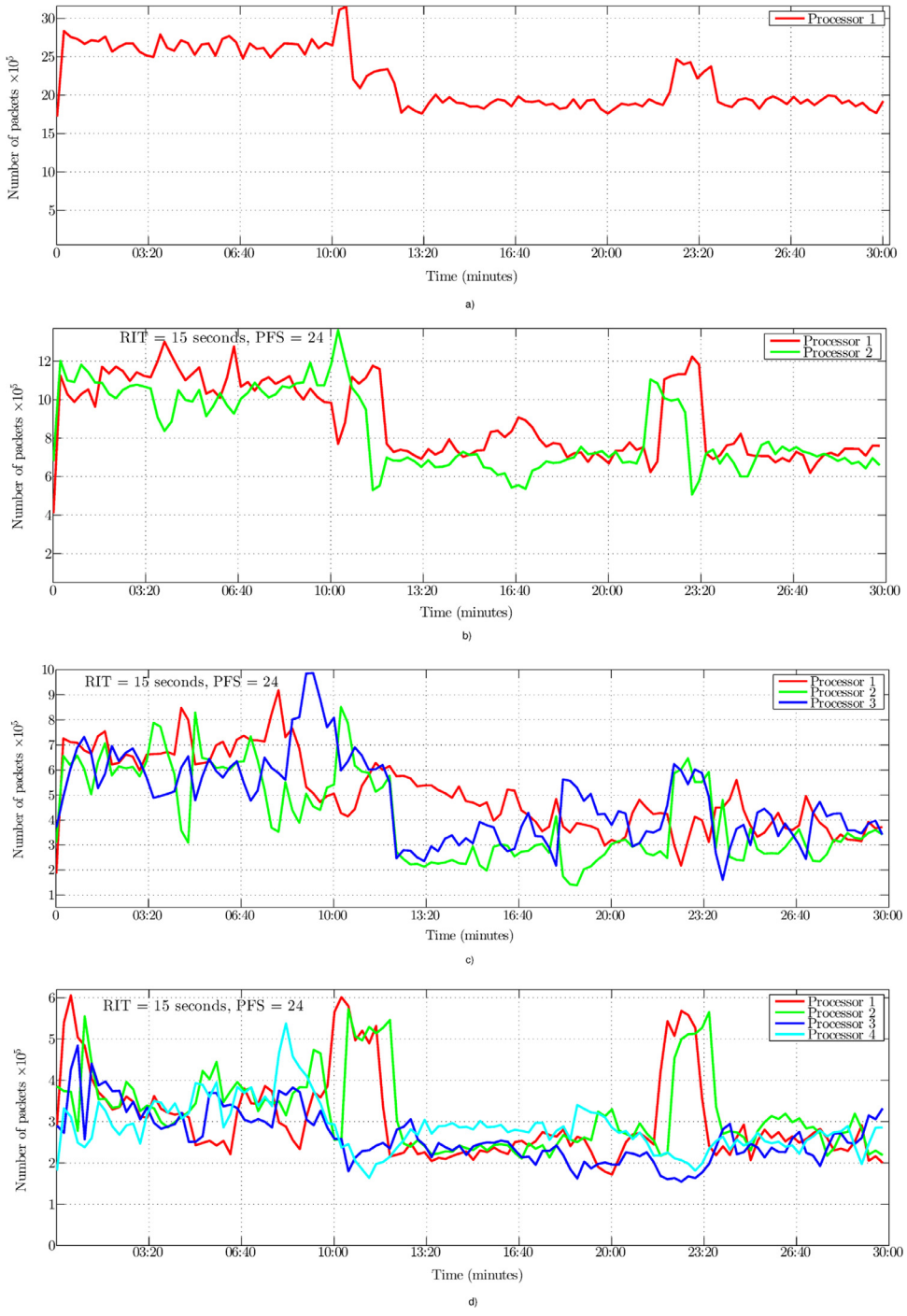

Fig. 4. Load distribution for setups with 1, 2, 3 and 4 processors. X axes show time intervals. Y axes denote the packet counts, with each tick denoting 100,000 packets.

through different processing nodes. If only load balancing is observed (without active attacks), a smaller value of the PFS parameter should yield better results in cases where the traffic involves a large number of prefixes in destination addresses evenly distributed across the entire address space. Larger values (i.e. smaller subnets) result in a greater number of rules on the switch, and more overhead for the controller during the recalculation and redistribution. This, in turn, may lead to longer reaction times, especially at larger speeds and if coupled with a larger RIT value. Experimental results in Section 5.3 have shown that the extreme values of PFS usually fail to achieve positive results is terms of load balancing.

If the effects of the PFS parameter are observed from the security standpoint, different values may also affect the distribution of the attack traffic. Namely, for larger PFS values (smaller subnets), there is a greater probability that two simultaneous attacks will be detected by different processors (i.e. there is a greater chance that the prefixes belonging to the same larger network will not be forwarded to the same processor), in which case the detection algorithm is performed independently on each of the nodes. In order to empirically prove this point, the attacks used in Section 5.3 have been purposefully configured to target separate processing nodes. The detection procedure benefits from this distribution by

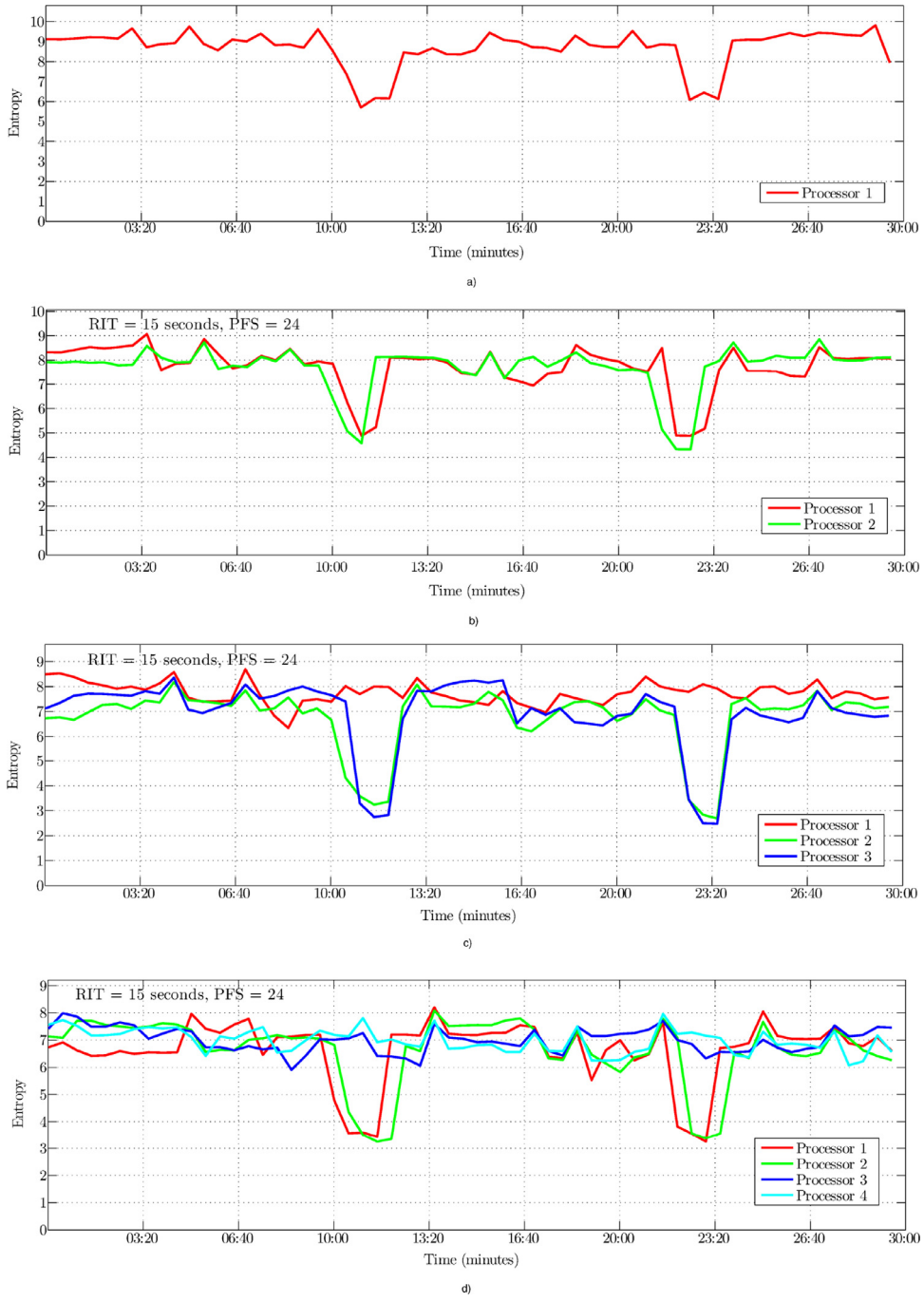

Fig. 5. Entropy values for setups with 1, 2, 3 and 4 processors. X axes show time intervals. Y axes denote the entropy values.

detecting a much bigger difference in entropy, and a smaller total load on each of the processors.

Several possible values have been tested for the Simpleweb dataset, and the PFS value of 24 produced the satisfactory results in all cases. In situations where the traffic distribution is more uniform, the setups with an even number of processors show a slightly better behaviour in terms of equalizing the load.

The length of the Keepalive interval has a significant effect on both the load balancing and the attack detection procedures. Shortening the Keepalive timer (i.e. the length of the time slice) would create a very responsive load balancing mechanism (as the rebalancing is initiated upon receipt of the keepalive messages from the processors), but would have a negative effect on the attack detection procedures (the shorter the interval, the more aggressive the attack has to be in order to be detected in a single time slice).

5.4. Discussion

The test results shown in Section 5.3 offer a comparison between a non-distributed and distributed mode of operation for TIDS, as well as some important conclusions regarding the values of parameters involved in the operation algorithm.

5.4.1. The evaluation of experimental results

As expected, the processor loads shown in Figs. 4a–d show a significant increase in the number of packets during the attacks. The throughput values measured at the processors, although

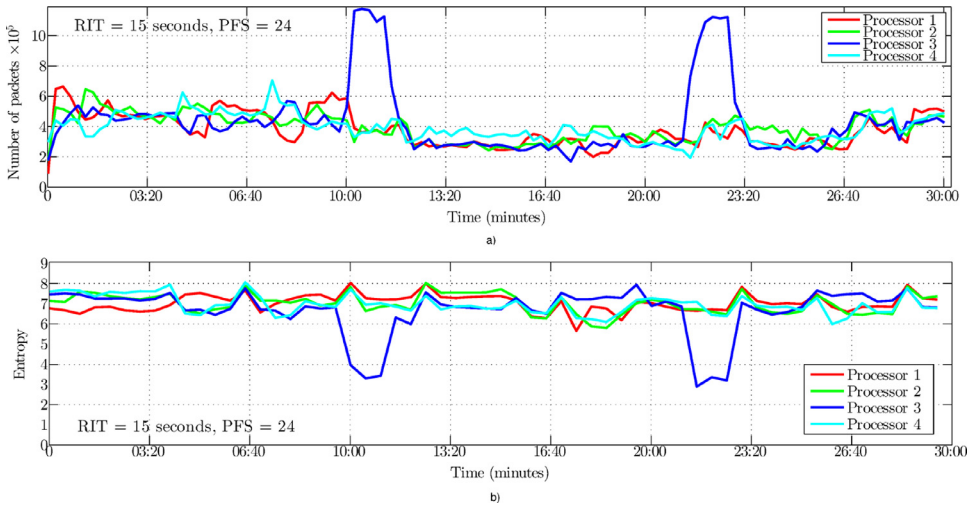

Fig. 6. Entropy and load values for a DDoS attack.

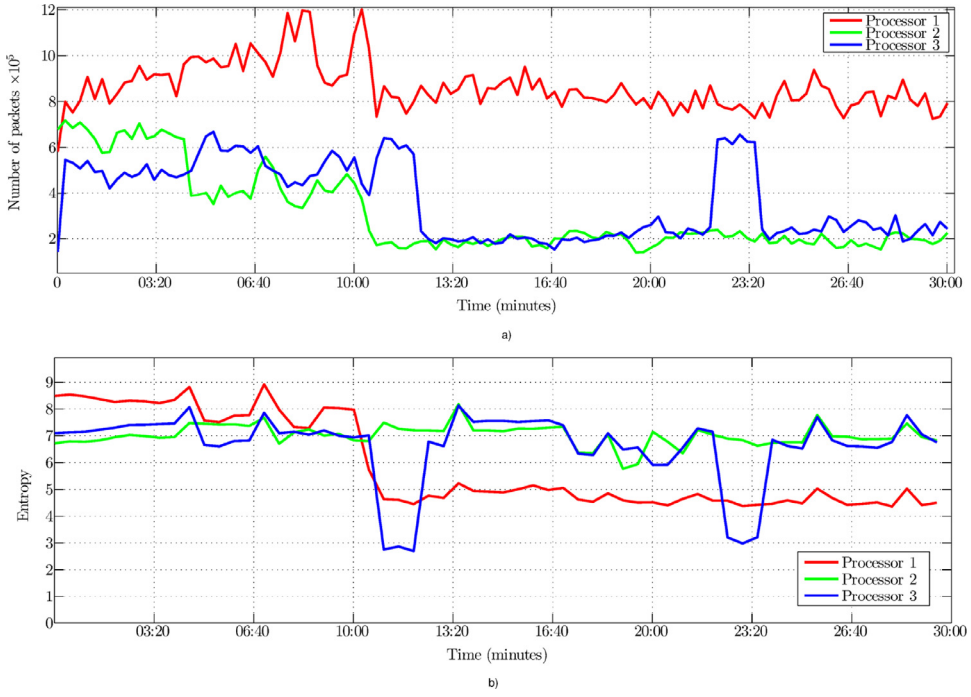

Fig. 7. Load and entropy values for the setup with 3 processors without the load balancing. X axes show time intervals.

not shown in this paper, confirm the same conclusion. The attack stream generators were active for the entire duration of the test, but the spikes in processor loads appear only twice: after the start of the initial attack, and again after the expiration of the first ban.

The load balancing mechanism equalizes the load on each of the processors, so that it is irrelevant which of the processors receives the attack traffic. If the load is not equal at the time the attack begins, the attack traffic might affect the entropy values differently for different processors, depending on the current address distribution. This is visible in Figs. 7a and b, especially when compared to corresponding Figs. 4c and 5c, which show the load and entropy for the same setup, but with the load balancing activated. Without the load balancing, one of the processors receives a significantly larger portion of the background traffic, which masks the attack traffic enough so that the change in entropy not sharp enough and is not detected as an attack. At the same time, the second attack (which has the same intensity as the first one) is processed by a processor with a lower portion of background traffic and is easily detected. With the load balancing, all processors have an equal chance of detecting an attack and the attack should have the same effect on the entropy value, regardless of the number of processors or its intensity. Furthermore, two consecutive attacks with the same source and destination may be processed by different processors, without the transition affecting the detection process.

5.4.2. The advantages of the entropy-based attack detection

Compared to an approach based on packet counts or traffic bit rate, the entropy-based approach offers an important advantage. During the normal network operation (especially in cloud environments and data centres with a large protected network), the packet count and traffic bit rate may fluctuate significantly even without an actual attack in progress. On the other hand, a change in the entropy occurs when the distribution of the packet destinations changes significantly, so by using an entropy-based approach the system is able to function with almost no previous knowledge

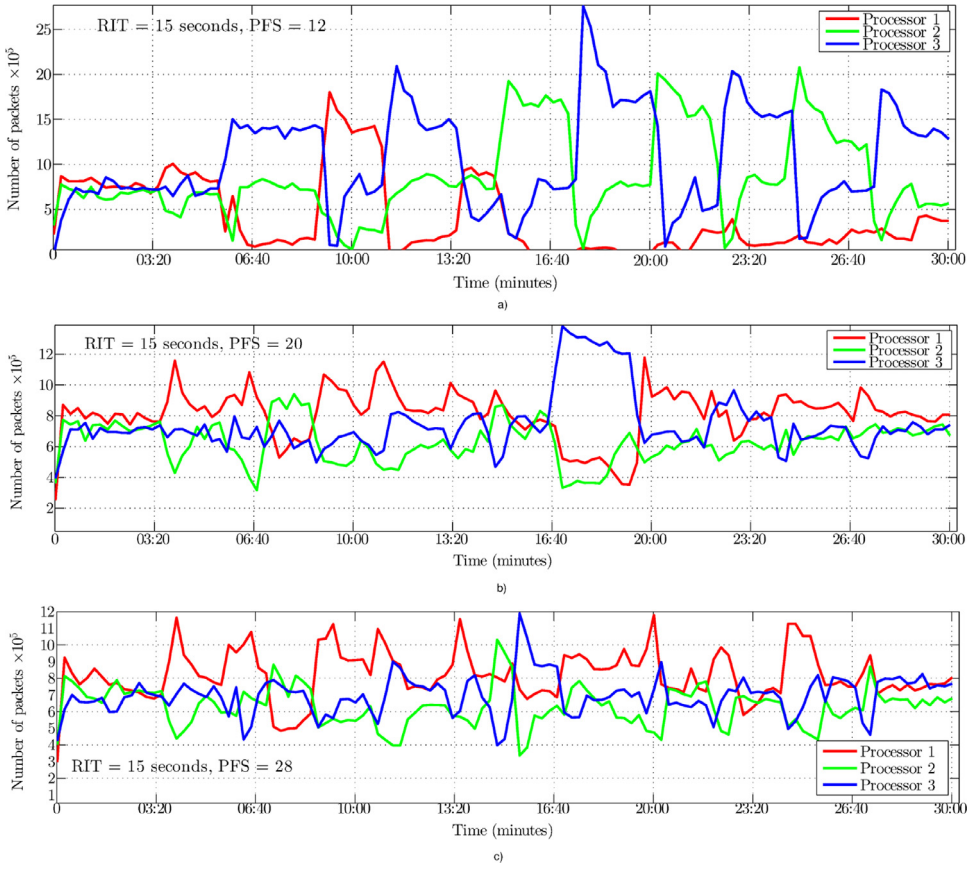

Fig. 8. The load balancing results for different values of the PFS parameter.

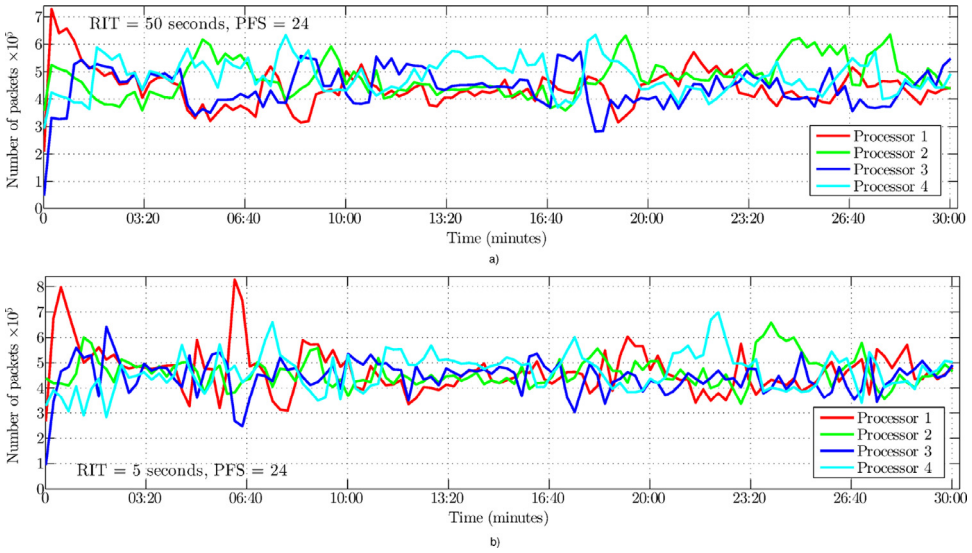

Fig. 9. The load balancing results for different values of the RIT parameter.

about the operation of the network or the traffic patterns (all required data is obtained during the learning period).

When discussing the process of attack detection, one of the most important factors is certainly the entropy difference between any two consecutive time slices required to indicate the start of an attack. Although test results display similar entropy patterns for both configurations, the multiple-processor setups show a much steeper slope of the entropy graph (i.e. the entropy is decreased in much shorter time) and a lower final entropy value in general after the start of the attack. Due to the load balancing, a processor does not receive traffic for all destinations within the protected network, so the attack traffic occupies a larger portion of the total number of processed packets on that processor. This, in turn, results in a lower entropy value compared to the one measured before the attack (and a faster change of that value). Conversely, the difference in entropy for the single-processor setup is much smaller. In order to detect the attack in all experimental setups, the entropy threshold parameter had to be configured to 80% of the entropy value

before the attack (the baseline value). However, the scenarios with 3 and 4 processors worked properly even with the entropy threshold set to a much lower value (60 %). Generally, the higher the value of the attack threshold, the more sensitive the system would be to entropy changes. The lower value gives the system additional stability and minimizes the risk of classifying normal traffic patterns as DDoS attacks. The final entropy value in a single-processor environment is low enough that there is no danger of misinterpreting the attack as valid traffic, but it is still significantly higher than with multiple processors. More to the point, from the results displayed in Figs. 5a–d, the final entropy value is inversely proportional to the number of active processors.

5.4.3. False positive detections

The problem of false positive detections is an important issue in the case of volumetric DoS. An occurrence in which a large number of legitimate users generate an enormous amount of traffic towards one or more hosts in an attempt to access interesting content is known as a flash crowd [49]. If the flash crowd occurs over a short period of time, it would manifest itself as a Distributed DoS attack due to the change in entropy in favour of the server hosting the interesting content. Several papers propose different ways of distinguishing flash crowds from ordinary DDoS attacks [33,49,50]. However, the proposed solutions rely on assumptions about the distribution of source prefixes [50] or the number of attackers [49], which can be used by the attackers to bypass such detection systems. On the other hand, Kandula et. al [33] argue that the flash crowd cannot be effectively distinguished from a DDoS attack by analysing the traffic patterns or source distributions. They propose an efficient detection system, but limit its scope to HTTP protocol using the features of the protocol itself to isolate compromised hosts from the legitimate users. When observed from a standpoint of a network-layer device (as is the case with TIDS), it is not feasible to develop a solution that would perform across all application-layer protocols with equal efficiency. Therefore, the question remains whether a large amount of traffic should be considered legitimate if the final effect is the same as with a DDoS attack (i.e. rendering a service unavailable), and whether the classification of such an occurrence as an attack should be considered a false positive detection.

6. Conclusion

This paper described a system which monitors network traffic for denial of service attacks and is capable of preventing some of them. The system is fully transparent and can be inserted on any link in the network. It introduces minimal latency into the packet transfer as it is designed to be scalable through the implementation of efficient packet processing methods and efficient load balancing scheme over multiple traffic processors.

The current version of the system can easily be expanded to include detection capabilities for other types of network attacks which can loosely be classified as DoS attacks. Attacks induced by invalid TCP sequence numbers, SYN-floods and other similar attacks could be detected without major changes to the platform, and without additional memory or processing requirements. The detection algorithms used for these attacks are invulnerable to traffic redistribution and could be efficiently run on the proposed platform.

Currently, there is no effective way of isolating the attack sources during a DDoS attack. Implementations proposed in literature offer solutions that are theoretically effective, but require provisions that would be difficult to achieve in practice. Arguably, by blocking the destination host after a distributed attack is detected, the goal of the attack is partially fulfilled (the service becomes unavailable), but the infrastructure is preserved from further damages. Nevertheless, the process of isolating the compromised hosts involved in a DDoS attack is a possible subject for further research.

The test results shown in this paper have proved the feasibility of an SDN-based scalable solution for detecting DDoS attacks in real time. Applications for this type of security layer are numerous, especially with the increasing popularity of cloud infrastructures and high-speed network environments in general.

References

[1] S.W. Cadzow, Security mechanisms in converged networks, in: Proceedings of the First IEE International Conference on Commercialising Technology and Innovation, IET, 2005.

[2] N. Hoque, M.H. Bhuyan, R.C. Baishya, D.K. Bhattacharyya, J.K. Kalita, Network attacks: taxonomy, tools and systems, J. Netw. Comput. Appl. 40 (2014) 307–324, doi:10.1016/j.jnca.2013.08.001.

[3] J. Mirkovic, G. Prier, P. Reiher, Attacking DDoS at the source, in: Proceedings of 10th IEEE International Conference on Network Protocols, 2002, pp. 312–321, doi:10.1109/ICNP.2002.1181418.

[4] J. Steinberger, A. Sperotto, H. Baier, A. Pras, Collaborative attack mitigation and response: a survey, in: Proceedings of the 2015 IFIP/IEEE International Symposium on Integrated Network Management (IM), IEEE, 2015, pp. 910–913, doi:10.1109/INM.2015.7140407.

[5] S.T. Zargar, J.B. Joshi, D. Tipper, A survey of defense mechanisms against distributed denial of service (DDoS) flooding attacks, IEEE Commun. Surv. Tut. (2013) 2046–2069, doi:10.1109/SURV.2013.031413.00127.

[6] R. Hofstede, V. Bartos, A. Sperotto, A. Pras, Towards real-time intrusion detection for netflow and IPFIX, in: Proceedings of the 2013 9th International Conference on Network and Service Management (CNSM), IEEE, 2013, pp. 227–234, doi:10.1109/CNSM.2013.6727841.

[7] T. Fu, D. Zhang, L. Xia, M. Li, Ipfix ie extensions for DDoS attack detection, internet draft, 2015, (https://tools.ietf.org/html/draft-fu-dots-ipfix-extension-00).

[8] K. Wang, S.J. Stolfo, Anomalous payload-based network intrusion detection, in: Recent Advances in Intrusion Detection, 7th International Symposium, RAID 2004, Sophia Antipolis, France, vol. 3224, 2004, pp. 203–222, doi:10.1007/978-3-540-30143-1_11.

[9] J.J. Davis, A.J. Clark, Data preprocessing for anomaly based network intrusion detection: a review, Comput. Secur. 30 (6-7) (2011) 353–375, doi:10.1016/j.cose.2011.05.008.

[10] A. Lakhina, M. Crovella, C. Diot, Mining anomalies using traffic feature distributions, SIGCOMM Comput. Commun. Rev. 35 (4) (2005) 217–228, doi:10.1145/1090191.1080118.

[11] K. Giotis, C. Argyropoulos, G. Androulidakis, D. Kalogeras, V. Maglaris, Combining openflow and sflow for an effective and scalable anomaly detection and mitigation mechanism on sdn environments, Comput. Netw. 62 (2013) 122–136, doi:10.1016/j.bjp.2013.10.014.

[12] N. Handigol, S. Seetharaman, M. Flajslik, R. Johari, N. Mckeown, Aster∗x: load-balancing as a network primitive, in: Proceedings of the 9th GENI Engineering Conference, 2010.

[13] A. Dixit, F. Hao, S. Mukherjee, T. Lakshman, R. Kompella, Towards an elastic distributed SDN controller, in: Proceedings of the Second ACM SIGCOMM Workshop on Hot Topics in Software Defined Networking, ACM, 2013, pp. 7–12, doi:10.1145/2491185.2491193.

[14] H. Lai, S. Cai, H. Huang, J. Xie, H. Li, A parallel intrusion detection system for high-speed networks, in: Applied Cryptography and Network Security, Second International Conference, ACNS 2004, 2004, pp. 439–451, doi:10.1007/978-3-540-24852-1_32.

[15] K. Xinidis, I. Charitakis, S. Antonatos, K.G. Anagnostakis, E.P. Markatos, An active splitter architecture for intrusion detection and prevention, IEEE Trans. Depend. Secure Comput. 3 (2006) 31–44, doi:10.1109/TDSC.2006.6.

[16] J. Coppens, S.V. den Berghe, H. Bos, E.P. Markatos, F.D. Turck, A. Oslebo, S. Ubik, Scampi: a scalable and programmable architecture for monitoring gigabit networks, management of multimedia networks and services, in: Proceedings of 6th IFIP/IEEE International Conference, MMNS 2003, Belfast, Northern Ireland, UK, September 7-10, 2003., 2003, pp. 475–487, doi:10.1007/978-3-540-39404-4_36.

[17] R. Kanagavelu, B.S. Lee, R.F. Miguel, L.N.T. Dat, L.N. Mingjie, Software defined network based adaptive routing for data replication in data centers, in: Proceedings of 19th IEEE International Conference on Networks (ICON), 11-13 Dec. 2013, Singapore, IEEE, 2013, pp. 1–6, doi:10.1109/ICON.2013.6781967.

[18] S. Fang, Y. Yu, C.H. Foh, K.M.M. Aung, A loss-free multipathing solution for data center network using software-defined networking approach, IEEE Trans. Magn. 49 (2013) 2723–2730, doi:10.1109/TMAG.2013.2254703.

[19] A. Tavakoli, M. Casado, T. Koponen, S. Shenker, Applying NOX to the datacenter, in: Proceedings of workshop on Hot Topics in Networks (HotNets-VIII), 2010.

[20] T.D. Nadeau, K. Gray, SDN: software defined networks, OReilly Media, Inc., 2013.

[21] M.F. Bari, S.R. Chowdhury, R. Ahmed, R. Boutaba, D.R. Cheriton, Policycop: an autonomic QoS policy enforcement framework for software defined networks, in: Proceedings of IEEE SDN for Future Networks and Services (SDN4FNS), 11-13 Nov. 2013, Trento, IEEE, 2013, pp. 1–7, doi:10.1109/SDN4FNS.2013.6702548.

[22] S. Namal, I. Ahmad, A. Gurtov, M. Ylianttila, SDN based inter-technology load balancing leveraged by flow admission control, in: Proceedings of IEEE SDN for Future Networks and Services (SDN4FNS), 11-13 Nov. 2013, Trento, IEEE, 2013, pp. 1–5, doi:10.1109/SDN4FNS.2013.6702551.
[23] B. Nunes, M. Mendonca, N. Xuan-Nam, K. Obraczka, T. Turletti, A survey of software-defined networking: Past, present, and future of programmable networks, IEEE Commun. Surv. Tut. 16 (2014) 1617–1634, doi:10.1109/SURV.2014.012214.00180.
[24] A. Lara, A. Kolasani, B. Ramamurthy, Network innovation using openflow: a survey, IEEE Commun. Surv. Tut. 16 (2014) 493–512, doi:10.1109/SURV.2013.081313.00105.
[25] R. S. F.Community, Ryu SDN controller, 2014, (http://osrg.github.io/ryu/).
[26] L. Deri, S.P.A. Netikos, V.D.B. Km, L.L. Figuretta, Improving passive packet capture: beyond device polling, in: Proceedings of SANE 2004, 2004, pp. 85–93.10.1.1.58.3128
[27] Intel, Intel data plane development kit: programmer's guide, 2013, (http://www.intel.com/content/www/us/en/intelligent-systems/intel-technology/intel-dpdk-programmers-guide.html).
[28] Intel, DPDK performance report, 2013, (http://www.intel.com/content/www/us/en/intelligent-systems/intel-technology/intel-dpdk-programmers-guide.html).
[29] I. Corona, G. Giacinto, F. Roli, Adversarial attacks against intrusion detection systems: taxonomy, solutions and open issues, Inf. Sci. 239 (2013) 201–225, doi:10.1016/j.ins.2013.03.022.
[30] I.S. Association, Ethertype, 2014, (http://standards.ieee.org/develop/regauth/ethertype/eth.txt).
[31] R. de O. Schmidt, L. Hendriks, A. Pras, R. van der Pol, Openflow-based link dimensioning, Innovating the Network for Data Intensive Science Workshop, INDIS 2014, SCinet, 2014.
[32] Y. Rekhter, T. Li, S. Hares, A border gateway protocol 4 (BGP-4), 2006, (https://tools.ietf.org/html/rfc4271).
[33] S. Kandula, D. Katabi, M. Jacob, A. Berger, Botz-4-sale: Surviving organized ddos attacks that mimic flash crowds, in: Proceedings of the 2Nd Conference on Symposium on Networked Systems Design & Implementation - Volume 2, in: NSDI'05, USENIX Association, Berkeley, CA, USA, 2005, pp. 287–300.
[34] S.T. Chou, A. Stavrou, J. Ioannidis, A.D. Keromytis, gore: Routing-assisted defense against ddos attacks, in: J. Zhou, J. Lopez, R. Deng, F. Bao (Eds.), Information Security, Lecture Notes in Computer Science, 3650, Springer Berlin Heidelberg, 2005, pp. 179–193, doi:10.1007/11556992_13.
[35] C. Hopps, Analysis of an equal-cost multi-path algorithm, 2000, (https://tools.ietf.org/html/rfc2992).
[36] K. Kompella, J. Drake, S. Amante, W. Henderickx, The use of entropy labels in mpls forwarding, 2012, (https://tools.ietf.org/html/rfc6790).
[37] J. yeon Jo, Y. Kim, H.J. Chao, F. Merat, Internet traffic load balancing using dynamic hashing with flow volume, in: in Internet Performance and Control of Network Systems III at SPIE ITCOM 2002, 2002, pp. 154–165.
[38] S. Kandula, S. Sengupta, A. Greenberg, P. Patel, R. Chaiken, The nature of data center traffic: Measurements & analysis, in: Proceedings of the 9th ACM SIGCOMM Conference on Internet Measurement Conference, in: IMC 09, ACM, New York, NY, USA, 2009, pp. 202–208, doi:10.1145/1644893.1644918.
[39] R. Krishnan, L. Yong, A. Ghanwani, N. So, B. Khasnabish, Mechanisms for optimizing link aggregation group (LAG) and equal-cost multipath (ECMP) component link utilization in networks, 2015, (https://tools.ietf.org/html/rfc7424).
[40] Z.A. Qazi, C.-C. Tu, L. Chiang, R. Miao, V. Sekar, M. Yu, Simple-fying middlebox policy enforcement using sdn, in: Proceedings of the ACM SIGCOMM 2013 Conference on SIGCOMM, in: SIGCOMM '13, ACM, 2013, pp. 27–38, doi:10.1145/2486001.2486022.
[41] J.P. Ponciano, N. Anani, Load balancing in modern network infrastructures a simulation model, in: Proceedings of the 2014 9th International Symposium on Communication Systems, Networks & Digital Signal Processing (CSNDSP), IEEE, 2014, pp. 841–846, doi:10.1109/CSNDSP.2014.6923944.
[42] Y. Kaymak, R. Rojas-Cessa, Per-packet load balancing in data center networks, in: Proceedings of the 2015 36th IEEE Sarnoff Symposium, IEEE, 2015, pp. 140–144, doi:10.1109/SARNOF.2015.7324658.
[43] A.R. Curtis, J.C. Mogul, J. Tourrilhes, P. Yalagandula, P. Sharma, S. Banerjee, Devoflow: scaling flow management for high-performance networks, in: Proceedings of the ACM SIGCOMM 2011 Conference, in: SIGCOMM '11, ACM, 2011, pp. 254–265, doi:10.1145/2018436.2018466.
[44] VMware, Vmware virtual networking concepts, 2015, (http://www.vmware.com/files/pdf/virtual_networking_concepts.pdf).
[45] R. van Rijswijk-Deij, A. Sperotto, A. Pras, Dnssec and its potential for ddos attacks - a comprehensive measurement study, in: Proceedings of the Internet Measurement Conference 2014, ACM Press, Vancouver, BC, Canada, 2014.
[46] Tcpreply, 2015, (http://tcpreplay.appneta.com/).
[47] P. R.for the Defense of Infrastructure Against Cyber Threats (PREDICT), Predict data repository, 2015, (https://www.predict.org/Default.aspx?tabid=169).
[48] Bonesi - the DDoS botnet simulator, 2015, (https://code.google.com/p/bonesi/).
[49] K. Prasad, A. Reddy, K. Rao, Discriminating ddos attack traffic from flash crowds on internet threat monitors (ITM) using entropy variations, Afr. J. Comput. ICT 6 (2013) 53–62.
[50] K. Li, W. Zhou, P. Li, J. Hai, J. Liu, Distinguishing DDoS attacks from flash crowds using probability metrics, in: Proceedings of the Third International Conference on Network and System Security, 2009. (NSS '09.), IEEE, 2009, pp. 9–17, doi:10.1109/NSS.2009.35.

Ognjen Joldzic is a Teaching Assistant at the Department of Computer Science at the Faculty of Electrical Engineering of the University in Banja Luka, Bosnia and Herzegovina. He obtained his BSc in 2008, and his MSc in 2010 from the Faculty of Electrical Engineering in Banja Luka. He is currently a PhD candidate at the Department of Computer Science at the Faculty of Electrical Engineering in Banja Luka. He is also a member of the S-CUBE - Software Systems and Security Research Group. His research interests include network security, intrusion detection and prevention, software defined networking and software development.

Zoran Djuric is a Professor at the Faculty of Electrical Engineering at University of Banjaluka. He is also a coordinator of S-CUBE - Software Systems and Security Research Group. His research interests include computer systems security, cryptography, PKI, e-payment systems and protocols, formal verification, object-oriented programming and modeling, and service-oriented architectures. He has (co-)authored more than 60 research papers.

Pavle Vuletic received his BSc, MSc and PhD in Computer Systems and Network Architecture from the University of Belgrade, School of Electrical Engineering. He worked on the development of the Serbian Research and Education Network (AMRES). He is currently a deputy director of AMRES and an assistant professor at the University of Belgrade, School of Electrical Engineering at the Department of Computer Engineering and Information Theory, teaching an Advanced Computer Networks course. His research interests span from network performance, monitoring and modern network management principles to network security and traffic characterization.