On Application-Layer DDoS Attack Detection in High-Speed Encrypted Networks

[1]Mikhail Zolotukhin, [2]Tero Kokkonen, [3]Timo Hämäläinen, [4]Jarmo Siltanen

[1,2,3]*Department of Mathematical Information Technology, University of Jyväskylä*
Jyväskylä, Finland
{mikhail.m.zolotukhin, timo.t.hamalainen}@jyu.fi, tero.t.kokkonen@student.jyu.fi

[2,4]*Institute of Information Technology, JAMK University of Applied Sciences*
Jyväskylä, Finland
{tero.kokkonen, jarmo.siltanen}@jamk.fi

Abstract

Application-layer denial-of-service attacks have become a serious threat to modern high-speed computer networks and systems. Unlike network-layer attacks, application-layer attacks can be performed by using legitimate requests from legitimately connected network machines which makes these attacks undetectable for signature-based intrusion detection systems. Moreover, the attacks may utilize protocols that encrypt the data of network connections in the application layer making it even harder to detect attacker's activity without decrypting users network traffic and violating their privacy. In this paper, we present a method which allows us to timely detect various application-layer attacks against a computer network. We focus on detection of the attacks that utilize encrypted protocols by applying an anomaly-detection-based approach to statistics extracted from network packets. Since network traffic decryption can violate ethical norms and regulations on privacy, the detection method proposed analyzes network traffic without decryption. The method involves construction of a model of normal user behavior by analyzing conversations between a server and clients. The algorithm is self-adaptive and allows one to update the model every time when a new portion of network traffic data is available. Once the model has been built, it can be applied to detect various types of application-layer denial-of- service attacks. The proposed technique is evaluated with realistic end user network traffic generated in our virtual network environment. Evaluation results show that these attacks can be properly detected, while the number of false alarms remains very low.

Keywords*: Network Security, Intrusion Detection, Denial of Service, Anomaly Detection, Traffic Clustering*

1 Introduction

The Internet has become the major universal communication infrastructure. Unfortunately, it is also subject to cyber-attacks in growing numbers and varieties. Denial-of-service (DoS) attacks have become frighteningly common in modern high- speed networks. On a daily basis, hundreds of websites are hit by networks of infected machines which flood them with junk data making them inaccessible for regular users [1] [2] [3]. In general, DoS attacks aim to disable a computer or network system using lots of messages which need responses consuming the bandwidth or other resources of the system.

Since it is difficult for an attacker to overload the targets resources from a single computer, DoS attacks are often launched via a large number of distributed attacking hosts in the Internet. Such distributed DoS (DDoS) attacks can force the victim to significantly downgrade its service performance or even stop delivering any service [4]. Designed to elude detection by today's most popular tools, these attacks can quickly incapacitate a targeted business, costing victims in lost revenue and productivity.

Traditional DDoS attacks such as ICMP flooding and SYN flooding are carried out at the network layer. The purpose of these attacks is to consume the network bandwidth and deny service to legitimate users of the victim systems. This type of attack has been well studied recently and different schemes have been proposed to protect the network and equipment from such bandwidth attacks [5][6][7]. For this reason, attackers shift their offensive strategies to application-layer attacks. Application-layer DDoS attacks may focus on exhausting the server resources such as Sockets, CPU, memory, disk bandwidth, and I/O bandwidth. Unlike network-layer DoS attacks, application-layer attacks do not necessarily rely on inadequacies in the underlying protocols or operating systems. They can be performed by using legitimate requests from legitimately connected network machines. This makes application- layer DDoS attacks so difficult to detect and prevent.

Anomaly-based approach is a promising solution for detecting and preventing application-layer DDoS attacks. Such approach learns the features of event patterns which form normal behavior, and, by observing patterns that deviate from the established norms (anomalies), detects when an intrusion has occurred. Thus, systems which use the anomaly detection approach are modeled according to normal user-behavior and, therefore, are able to detect zero-day attacks, i.e. intrusions unseen previously. The problem of anomaly-based detection of application-layer DoS and DDoS attacks is of great interest nowadays [8][9][10][11][12][13][14][15] [16] [17].

In this study, we focus on the detection of the attacks that involve the use of HTTP protocol since it is the most prevalent application-layer denial-of-service attack type nowadays [2]. Study [10] divides HTTP-based DDoS attacks into three categories based on the level of their sophistication. The first category is trivial DDoS attacks, during which each bot participating in the attack sends one or a limited number of unrelated HTTP attacks towards the target site. This type of at- tacks includes such well-known attacks as Sslsqueeze [18] and Slowloris [8]. The second category includes attacks that are carried out by bots generating random sequences of browser- like requests of web-pages with all of their embedded content making the attack traffic indistinguishable from the regular human traffic. The last category contains more advanced DoS attacks that are predicted to rise in popularity in the future. These advanced attacks will consist of sequences of HTTP requests which are carefully chosen so as to better mimic the browsing behavior of regular human users.

Despite the rising interest to the detection of application- layer DDoS attacks utilizing HTTP protocol, most of the current researches concentrate on the analysis of information extracted from network packet payload which includes web resource requested, request method, session ID and other parameters. However, nowadays many DDoS attacks are utilizing secure protocols such as SSL/TLS that encrypt the data of network connections in the application layer which makes it impossible to detect attacker activity based on the analysis of packets' payload without decrypting it [19]. For this reason, the detection of DDoS attacks is supposed to be carried out with the help of statistics that can be extracted mostly from network packet headers.

Another challenge in the problem of application-layer DDoS attacks detection is distinguishing these attacks from flash crowds. Flash crowds are large surges of legitimate traffic which occur on popular web sites when thousands of requests access to the web servers over the relatively short period of time. Flash crowds are quite similar with DDoS attacks in terms of network anomaly and traffic phenomenon. They even can cause a web site or target to slow down its service for users or even temporarily shut down due to the significant increase of traffic [20].

It is also critical to implement a DDoS detection system which is capable to function effectively in computer networks that have high traffic and high-speed connectivity [21]. Moreover, since the mitigation of damage from a DDoS attack relies on its timely detection, the detection process is supposed to take place in an online mode. Nowadays, high-speed computer networks and systems deal with thousands traffic flows per second resulting in the data rate of several Gbps. For this reason, the construction of the normal user behavior model and the detection of an anomalous activity in a high-speed network requires considerable amounts of memory and computing resources. Thus, the problem of proper management of these resources is one of the most important challenges when designing a DDoS detection and prevention system.

In this study, we propose an application-layer DDoS attack detection scheme that meets the requirements discussed above. First, our method relies on the extraction of normal user behavior patterns and detection of anomalies that significantly deviate from these patterns. This allows us to detect even attacks that are carried out with legitimate requests from legitimately connected network machines. Moreover, the scheme proposed operates with information extracted from packet headers and therefore can be applied in secure protocols that encrypt the data of network connections without its decrypting. Finally, the normal user behavior model is obtained with the help of a data stream clustering algorithm that allows us to continuously update the model within memory and time restrictions. In order to evaluate our scheme, we implement a DDoS detection system prototype that employs the algorithm proposed. In addition, we create a virtual network environment that allows us to generate some realistic end user network traffic and different sorts of DDoS attacks. Simulation results show that these attacks can be properly detected, while the number of false alarms remains very low.

The rest of the paper is organized as follows. Problem formulation and related work are discussed in Section 2. Extraction of the most relevant features from network traffic is considered in Section 3. Section 4 describes our approach of DDoS attacks detection in encrypted network traffic. The implementation of this approach in the context of high-speed networks is presented in Section 5. In Section 6, we evaluate the performance of the technique proposed. Finally, Section 7 draws the conclusions and outlines future work.

2 Problem formulation

We concentrate on the detection of application-layer DDoS attacks in SSL/TLS traffic transferred over TCP protocol as the most popular reliable stream transport protocol. We consider a network system that consists of several web servers that provide various services to their end users by utilizing this protocol. Outgoing and incoming traffic of these servers is captured and analyzed in order to detect and prevent potential attacks. The analysis process can be divided into two main phases: training and detection. During the training phase, we aim to investigate the traffic and discover behavior patterns of normal users. It is assumed that the most part of the traffic captured during this training phase is legitimate. In real world, this can be achieved by filtering the traffic with the help of a signature-based intrusion detection system. Once normal user behavior patterns have been discovered, these patterns can be used to analyze network traffic and detect DDoS attacks against the servers in online mode.

3 Related work

The anomaly-based detection approach is often applied for the application-layer DDoS attack detection. Study [15] proposes an advanced entropy-based scheme, that analyzes the distribution of captured packets among network flows. The method allows to detect variable rate DDoS attacks, divide them into different fields and treat each field with different methods. Paper [14] proposes to cluster user sessions based on number of requests in the session, request rate, average popularity of objects in the session and average transition probability of objects in the session. Hierarchical clustering is applied to separate attack sessions from normal ones. Paper [9] shows a novel detection technique against HTTP-GET attacks that relies on clustering of user sessions based on minimizing entropy of requests in the sessions belonging to the same cluster. Bayes factor analysis is then used to detect attacking hosts. Paper [8] considers detection of slow DoS attacks by analyzing numbers of packets received by a web server over small time horizons. The detection is carried out by using two spectral metrics: average of these packet numbers and the mutual information of the fast Fourier transform applied to the number of packets received at the current time horizon and at the previous one. In [11], authors model a normal user browsing behavior by constructing a random walk graph based on sequences of web pages requested by each user. Once the random walk model has been trained, the users subsequent page request sequence is predicted based on page transition probabilities calculated. After this, an attacker can be detected by calculating the similarity between the predicted page request sequence and observed sequence in the subsequent observation period. Study [10] proposes the next-generation system for application-layer DDoS defense by modeling network traffic to dynamic web-domains as a data stream with concept drift. An outlier detection is carried out by using the normalized length of the longest common subsequence similarity metric applied to chronological browsing sequences visited Web pages during a user session. In [12], different features are constructed for each user session to differentiate between an attacker and a normal user. These features include the number of HTTP requests made by a user in a particular time slot, the number of unique URL requests, numbers of successful, redirected and invalid requests, and several others. Once all necessary features are calculated, logistic regression is used for modeling a normal user browsing behavior for detecting the application layer DDoS attack traffic. Study [13] proposes to search for abstract features for each user session with the help of a stacked auto encoder. After that, a logistic regression classifier is used to find network traffic related to a DDoS attack.

As one can notice, all of these studies propose to detect application-layer DDoS attacks by monitoring network packet payload. However, it remains unclear how to detect attacks in encrypted traffic. In this study, this problem is solved by modeling normal user behavior based on clustering feature vectors extracted from packet headers. Traffic clustering without using packet payload information is a crucial domain of research nowadays due to the rise in applications that are either encrypted or tend to change port consecutively. For example, study [22] uses k-means and model-based hierarchical clustering based on the maximum likelihood in order to group together network flows with similar characteristics. In [23], authors classify encrypted traffic with supervised learning algorithm C4.5 that generates a decision tree using information gain, semi-supervised k-means and unsupervised multi-objective genetic algorithm (MOGA).

In modern high-speed networks, large amounts of flow data are generated continuously at an extremely rapid rate. For this reason, it is not possible to store all the data in memory, which makes algorithms such as k-means and other batch clustering algorithms inapplicable to the problem of traffic flow clustering [24].

Data stream clustering algorithms is probably the most promising solution for this problem. Most stream clustering algorithms summarize the data stream using special data structures: cluster features, core sets, or grids. After performing the data summarization step, data stream clustering algorithms obtain a data partition via an offline clustering step [25]. Cluster feature trees and micro-cluster trees are employed to summarize the data in such algorithms as BIRCH [26], CluStream [27] and ClusTree [28]. The StreamKM++ algorithm [29] summarizes the data stream into a tree of coresets that are weighted subsets of points that approximate the input data. Grid-based algorithms such as DStream [30] and DGClust [31] partition the feature space into grid cells, each of which is representing a cluster. Approximate clustering algorithms such as streaming k-means [32] first choose a subset of the points from the stream, ensuring that the selected points are as distant from each other as possible, and then execute k-means on the data subset. Approximate stream kernel k-means algorithm [24] uses importance sampling to sample a subset of the data stream, and clusters the entire stream based on each data points similarity to the sampled data points in real-time.

4 Feature extraction

In order to extract features that are necessary for building a normal user behavior model and detecting outliers, we consider a portion of network traffic transferred in the computer system under inspection in some very short time window. The length of this time window should be picked in such a way that allows one to detect attacks timely.

The method proposed in this study is based on the analysis of network traffic flows. A flow is a group of IP packets with some common properties passing a monitoring point in a specified time interval. These common properties include transport protocol, the IP address and port of the source and IP address and port of the destination. As it was mentioned in the previous section, in this study, we concentrate on the traffic transferred over TCP. The time interval is considered to be equal to the time window defined previously. Moreover, when analyzing a traffic flow extracted in the current time window, we take into account all packets of this flow transferred during previous time windows. Resulting flow measurements provide us an aggregated view of traffic information and drastically reduce the amount of data to be analyzed. After that, two flows such as the source socket of one of these flows is equal to the destination socket of another flow and vice versa are found and combined together. This combination is considered as one conversation between a client and a server.

A conversation can be characterized by following four parameters: source IP address, source port, destination IP address and destination port. For each such conversation at each time interval, we extract the following information:

1) duration of the conversation
2) number of packets sent in 1 second
3) number of bytes sent in 1 second

4) maximal, minimal and average packet size
5) maximal, minimal and average size of TCP window
6) maximal, minimal and average time to live (TTL)
7) percentage of packets with different TCP flags: URG, ACK, PSH, RST, SYN and FIN
8) percentage of encrypted packets with different properties: handshake, alert, etc.

Features of types 2–8 are extracted separately for packets sent from the client to the server and from the server to the client. Features of types 2–7 are extracted from packet headers whereas a value of feature 8 can be found in packet payload even though it is encrypted.

It is worth to mention that here we do not take into account time intervals between subsequent packets of the same flow. Despite the fact, that increasing of these time intervals is a good sign of a DDoS attack, taking them into consideration leads to the significant increasing of the number of false alarms. It is caused by the fact, that when the server is under attack it cannot reply to legitimate clients timely as well, and, therefore, legitimate clients look like attackers from this point of view.

Values of the extracted feature vectors can have different scales. In order to standardize the feature vectors, max-min normalization is used. Max-min normalization performs a linear alteration on the original data so that the values are normalized within the given range. For the sake of simplicity, we map vectors to range [0,1]. Since all network traffic captured during the training stage is assumed to be legitimate, all the resulting standardized feature vectors can be used to reveal normal user behavior patterns and detect behavioral anomalies.

To map a value x_{ij} of the j-th attribute with values $(x_{1j}, x_{2j}, \dots)$ from range $x_{ij} \in [x_{min,j} x_{max,j}]$ to range $z_{ij} \in [0, 1]$ the computation is carried out as follows:

$$z_{ij} = \frac{x_{ij} - x_{min,j}}{x_{max,j} - x_{min,j}}. \quad (1)$$

5 Detection approach

In order to be able to classify application-layer DDoS attacks, we propose an anomaly-detection-based system that relies on the extraction of normal behavioral patterns during the training followed by the detection of samples that significantly deviate from these patterns. For this purpose, we analyze the traffic captured in the network under inspection with the help of several data mining techniques.

5.1 Training

Once all relevant features have been extracted and standardized, we divide resulting feature vectors into several groups by applying a clustering algorithm. Each such group is supposed to consist of objects that are in some way similar between themselves and dissimilar to objects of other groups. Clustering allows us to discover hidden patterns presented in the dataset to represent a data structure in an unsupervised way. There are many different clustering algorithms including hierarchical clustering algorithms, centroid-based clustering algorithms and density-based clustering algorithms. Each cluster calculated represents a specific class of traffic in the network system under inspection. For example, one such class can include conversations between a web server and clients which request the same web page of this server. Since the traffic may be encrypted it is not always possible to define what web page these clients request. However, since it is assumed that traffic being clustered is mostly legitimate, we can state that each cluster describes a normal user behavior pattern.

After that, we group all conversations which are extracted in certain time interval and have the same source IP address, destination IP address and destination port together and analyze each such group separately. Such approach is in-line with other studies devoted to the problem of application-based DDoS attacks detection [9][11][14]. Those studies analyze sequences of conversations (requests) belonging to one HTTP session. In our case, since the session ID cannot be extracted from encrypted payload, we focus on conversations initiated by one client to the destination socket during some short time interval. We can interpret a group of such conversations as a rough approximation of the user session.

For each such group of conversations, we obtain a sequence of numbers that are labels of the clusters found to which these conversations belong. An n-gram model is applied to extract new features from each such sequence. An n-gram is a sub-sequence of n overlapping items (characters, letters, words, etc.) from a given sequence. The n-gram models are used in speech recognition [33] and language processing [34]. Thus, the n-gram model transforms each user session to a sequence of n-labels. Then the frequency vector is built by counting the number of occurrences of each n-label in the analyzed session. The length of the frequency vector is k^n, where k is the number of conversation clusters.

Once new feature vectors have been extracted, a clustering algorithm can be applied to divide these vectors into groups. Similarly, to the clusters of conversations, each cluster of n-gram vectors represents a specific class of traffic in the network under inspection. For example, one such cluster can include clients that use some web service in similar manner. As previously, we consider each resulting cluster as a normal user behavior pattern, because it is assumed that traffic captured during the training is mostly legitimate. Thus, the normal user behavior model consists of the clusters of two types: clusters of conversations and clusters of user sessions, that directly depend on the conversation clusters.

5.2 Detection

Once the training has been completed, the system is able to detect network intrusions. To detect a trivial DoS attack we extract necessary features from a new conversation and classify the resulting feature vector according to the clusters found. If this vector does not belong to any of the clusters, the corresponding conversation is labeled as intrusive and it is supposed to be blocked by the server.

For example, for centroid-based clustering methods, to define whether a new vector belongs to a cluster or not, we calculate the distance between this vector and the cluster center. If the distance between the new vector and the cluster center is greater than a predefined threshold, this vector does not belong to the cluster. This threshold T for some cluster can be calculated based on vectors of the training set which belong to this cluster: $T = \mu + \gamma\sigma$, where μ is the average distance between the center and vectors of this cluster, σ is the standard deviation of these distance values and γ is some numeric parameter tuned during the detection system validation.

The anomalous conversations found allow us to detect trivial DDoS attacks. However, if the attacker is able to mimic the browsing behavior of a regular human user, conversations related to this attack might belong to one of the clusters of the normal behavior model and, therefore, remain undetected. In this case, n-gram statistics should be taken into consideration. Vectors obtained with n-gram model during an attack can differ markedly from vectors corresponding to legitimate traffic. Thus, we can define whether a computer or network system is under attack during the current time interval, and, moreover, find clients responsible for initiating conversations related to the attack.

Let us consider a client which initiates several connections of certain type during the recent time interval. After we classify these connections according to clusters of conversations obtained during the training, the n-gram model is applied to transform this client session into new feature vector. If this vector does not belong to any of the session clusters extracted during the training, then this new vector is classified as an anomaly and all connections of the client are considered as an attack. As one can see, in this case, we cannot define which connections of the client are normal and which connections have bad intent. However, this scheme allows us to find the attacker and what web service he attempts to attack. After that, the attacker can be black-listed and a more sophisticated approach can be applied to analyze the conversations initiated by this attacker in more details.

6 Implementation

In this section, we discuss how the approach described above can be implemented to protect a web service in a high-speed encrypted network. The most challenging part of the implementation is related to the training stage, since there can be huge volumes of network traffic generated continuously at an extremely rapid rate and there is no possibility to store all features extracted from this traffic in memory. For this reason, a data stream clustering algorithm can be applied to conversations between users and the web service. However, in this case, conversation clusters may change every time a new portion of traffic has arrived. In turn, this leads to modifications of n-gram vectors representing user sessions calculated during previous time windows. These modifications are supposed to be made before session clusters are updated with new data extracted from this portion of the traffic.

6.1 Clustering conversations

Let us consider conversations between clients and the web service that take place in the current time window. We propose to cluster feature vectors extracted from these conversations by constructing an array of centroids that summarizes the data partition [35] [36].

We consider feature vectors $X^t = \{x_1^t, \dots, x_c^t\}$ extracted from n_c^t conversations during the t-th time window and standardized vectors $Z^t = \{z_1^t, \dots, z_c^t\}$ that are obtained from raw vectors X^t with the help of max-min standardization using values $x_{max,j}^t$ and $x_{min,j}^t$.In order to find k centroids for these vectors, we can apply standard iterative refinement technique. First, k centroids are supposed to be initiated. It can be carried out by randomly choosing k feature vectors from Z^t. However, there is a more efficient way to select initial centroids by following kmeans++ procedure [37] [38]:

1) Choose an initial center m_1^t centroid uniformly at random from Z^t.
2) Choose the next center m_i^t, selecting $m_i^t = z_j^t \in Z^t$ with probability v_j^t:

$$v_j^t = \frac{min_{l \in \{1,\dots,i-1\}} d(m_l^t, z_j^t)}{\sum_{h=1}^{n_c^t} min_{l \in \{1,\dots,i-1\}} d(m_l^t, z_h^t)}, \tag{2}$$

where $j = \{1, \dots, n_c^t\}$.

3) Repeat the previous step $k - 1$ times to select k initial centroids.

Once initial centroids $m_1^t, \dots, m_k^t$ have been selected, we iteratively update the centroids as follows:

1) Assign each feature vector to the nearest centroid:

$$p_i^t = \{z \in Z^t : d(z, m_i^t) = \min_l d(z, m_l^t)\}. \tag{3}$$

2) For each partition p_i^t, find a new centroid:

$$m_i^t = \frac{1}{|p_i^t|} \sum_{z \in p_i^t} z. \tag{4}$$

These two steps are repeated until there are no longer changes in partitions during the assignment step.

Let us denote the raw vector that corresponds to standardized vector z as $x(z)$. For each resulting partition p_i^t, we store in memory its centroid

$$\mu_i^t = \frac{1}{|p_i^t(z)|} \sum_{z \in p_i^t(z)} x(z), \tag{5}$$

the number of feature vectors contained in partition p_i^t

$$w(p_i^t) = |\, p_i^t|, \tag{6}$$

and sum of squared features in these vectors

$$\varsigma(p_i^t) = \sum_{z \in p_i^t} x^2(z), \tag{7}$$

where $x^2(z) = (x^2(z_1), x^2(z_2), \dots)$. It is worth noting that despite we use standardized vectors for clustering, we store statistics calculated for raw vectors. In addition, we store vectors $x_{min,j}^t$ and $x_{max,j}^t$ used for the standardization.

We calculate all the partitions for τ consecutive time windows $t \in \{1, 2, \ldots, \tau\}$, where value of τ is defined by the memory constraints. In order to compress $\tau \times k$ resulting partitions into new k clusters. first, we calculate minimal $\bar{x}^{\tau}_{min,j}$ and maximal $\bar{x}^{\tau}_{max,j}$ feature values:

$$\begin{aligned} \bar{x}^{\tau}_{min,j} &= \min_{t\epsilon\{1,2,\ldots,\tau\}} x^{t}_{min,j}, \\ \bar{x}^{\tau}_{max,j} &= \max_{t\epsilon\{1,2,\ldots,\tau\}} x^{t}_{min,j}. \end{aligned} \tag{8}$$

These values are used to standardize centroids μ^t_i into vectors m^t_i for $t \in \{1,2,\ldots,\tau\}$ and $i\epsilon\{1,2,\ldots,k\}$.

We obtain new k centroids with the technique similar to the one described above. The only difference is that we take into account the numbers of feature vectors assigned with each centroid:

1) Choose an initial center $\bar{m}^{\tau}_1$ centroid uniformly at random from m^t_j where $t \in \{1,2,\ldots,\tau\}$ and $j\epsilon\{1,2,\ldots,k\}$.
2) Choose the next center $\bar{m}^{\tau}_j$, selecting $\bar{m}^{\tau}_j = m^t_j$ with probability v^t_j:

$$v^t_j = \frac{w(p^t_j) min_{l\in\{1,..,i-1\}} d(\bar{m}^{\tau}_l, m^t_j)}{\sum_{t=1}^{\tau}\sum_{h=1}^{k} w(p^t_h) min_{l\epsilon\{1,\ldots,i-1\}} d(\bar{m}^{\tau}_l, m^t_h)}. \tag{9}$$

3) Repeat the previous step $k-1$ times to select k initial centroids.

After that, new centroids are updated as follows:

1) Assign each old centroid m^t_j to the nearest new centroid $\bar{m}^{\tau}_i$:

$$\bar{p}^{\tau}_i = \{m^t_j : d(m^t_j, \bar{m}^{\tau}_i) = \min_l d(m^t_j, \bar{m}^{\tau}_l)\}. \tag{10}$$

2) For each partition $\bar{p}^{\tau}_i$, find a new centroid:

$$\bar{m}^{\tau}_i = \frac{1}{\sum_{m(z)\epsilon\bar{p}^{\tau}_i} w(z)} \sum_{m(z)\epsilon\bar{p}^{\tau}_i} w(z)z. \tag{11}$$

These two steps are again repeated until there are no longer changes in partitions during the assignment step.

For each resulting partition $\bar{p}^{\tau}_i$, we store in memory its centroid

$$\bar{\mu}^{\tau}_i = \frac{1}{\sum_{m(z)\epsilon\bar{p}^{\tau}} w(z)} \sum_{m(z)\epsilon\bar{p}^{\tau}_i} w(z)x(z), \tag{12}$$

the number of feature vectors associated with this centroid

$$w(\bar{p}^{\tau}_i) = \sum_{m(x)\epsilon\bar{p}^{\tau}_i} w(x), \tag{13}$$

and sum of squared features in these vectors

$$\varsigma(\bar{p}^{\tau}_i) = \sum_{m(x)\epsilon\bar{p}^{\tau}_i} \varsigma(x). \tag{14}$$

In addition, we substitute values $x^t_{min,j}$ and $x^t_{max,j}$ for $t \in \{1,2,\ldots,\tau\}$ with vectors $\bar{x}^{\tau}_{min,j}$ and $\bar{x}^{\tau}_{max,j}$ used for the standardization.

Once $\tau \times k$ partitions p^t_i have been compressed to new k partitions $\bar{p}^{\tau}_i$, information about the old partitions can be removed from the memory. After that, the algorithm continues in the same manner with finding partitions for the next $\tau - 1$ time windows $t \in \{\tau+1, \tau+2, \ldots, 2\tau-1\}$ and combining them with partitions $\bar{p}^{\tau}_i$ to obtain new k partitions.

In order to reduce the amount of computing resources required, this clustering procedure can be substituted with streaming k-means approximation proposed in [38]. However, from our numerical simulations, we notice, that the clustering algorithm used converges in just few iterations. To guarantee that the clustering is completed during the current time window, the number of iterations is recommended to be artificially limited.

6.2 Clustering sessions

We consider a group of conversations that are extracted at time window t and have the same source IP address, destination IP address and destination port. As mentioned in the Section 4 we interpret this group as a rough approximation of the user session. Once all connections at this time window have been divided into k partitions, for each user session, we obtain an n-gram vector of size k^n. Let us denote the new feature matrix as $Y^t = \{y_1^t, \dots, y_{n_s^t}^t\}$, where n_s^t is the number of different sessions at time window t.

We apply the same partition algorithm to new feature vectors in order to obtain K session centroids. As previously, for each resulting partition P_i^t, in addition to its centroid $M_i^t = m(P_i^t)$, we store in memory the number of feature vectors associated with this centroid:

$$w(P_i^t) = |P_i^t|. \tag{15}$$

However, instead of just sums of squared features in vectors assigned to a centroid, we calculate and store matrix $S(P_i^t)$ of size $k^n \times k^n$, the (j,l)-th element $S_{jl}(P_i^t)$ of which is calculated as follows:

$$S_{jl}(P_i^t) = \sum_{y \in P_i^t} y_j y_l, \tag{16}$$

where $j, l \in \{1, \dots, k^n\}$.

Once connection partitions for τ consecutive time windows $t \in \{1, 2, \dots, \tau\}$ have been calculated and $\tau \times k$ resulting partitions p_i^t have been compressed to new k connection clusters $\bar{p}_i^\tau$, information stored for each session partition P_i^t is supposed to be updated. It is caused by the fact that connection clusters have been changed which leads to the modifications in all the n-gram vectors. For this purpose, we introduce function $f(j, p_l^t, \bar{p}_i^\tau)$ with $j \in \{1, \dots, k^n\}$, such that if the j-th n-gram contains label l and partition p_l^t is assigned to new partition $\bar{p}_i^\tau$, the function returns the index that corresponds to the n-gram which is obtained from the j-th gram by substituting label l with label i.

Let us assume that conversation partition p_i^τ contains some of the partitions obtained at time window τ:

$$p_{i_1}^t, \dots, p_{i_q}^t \in \bar{p}_i^\tau. \tag{17}$$

It is worth noting that partition $\bar{p}_i^\tau$ can also contain partitions from other time windows, but they do not affect vectors Y^t at this point.

If the j-th gram contains label i, the j-th component of the i-th session centroid M_i^t is modified as follows:

$$M_{ij}^t = \sum_{a=1}^{q} M_{i,f\left(j, p_{i_a}^t, \bar{p}_i^\tau\right)}^t. \tag{18}$$

Similarly, elements of matrix $S(P_i^t)$ are modified:

$$\begin{aligned} S_{jl}(P_i^t) &= \sum_{a=1}^{q} S_{f\left(j, p_{i_a}^t, \bar{p}_i^\tau\right), l}(P_i^t), \qquad l \in \{1, \dots, k^n\}, \\ S_{lj}(P_i^t) &= \sum_{a=1}^{q} S_{l, f\left(j, p_{i_a}^t, \bar{p}_i^\tau\right)}(P_i^t), \qquad l \in \{1, \dots, k^n\}. \end{aligned} \tag{19}$$

If the j-th gram contains label $i_a \in \{1, \dots, i_q\}$ and does not contain label i, the j-th component of the i-th session centroid M_i^t and elements of matrix $S(P_i^t)$ become equal to zero:

$$M_{ij}^t = 0 \tag{20}$$

and

$$S_{jl}(P_i^t) = 0, l \in \{1, \dots, k^n\},$$
$$S_{lj}(P_i^t) = 0, l \in \{1, \dots, k^n\}. \tag{21}$$

The rest of the components do not change. Thus, we update information about session partitions to represent modifications in n-grams caused by the compression of connection clusters.

Once session partitions P_i^t, where $t \in \{1,\dots,\tau\}$ and $i \in \{1,\dots,K\}$, have been updated, these $\tau \times K$ partitions are compressed to new K clusters with the same technique as the one was applied for connections. For each resulting partition $\bar{P}_i^\tau$, we store in memory its centroid

$$m(\bar{P}_i^\tau) = \bar{M}_i^\tau, \tag{22}$$

the number of feature vectors associated with this centroid

$$w(\bar{P}_i^\tau) = \sum_{m(x)\in\bar{P}_i^\tau} w(x), \tag{23}$$

and matrix $S(\bar{P}_i^\tau)$, the (j,l)-th elements $S_{jl}(\bar{P}_i^\tau)$ of which is defined as follows:

$$S_{jl}(\bar{P}_i^\tau) = \sum_{m(x)\in\bar{P}_i^\tau} S_{jl}(x). \tag{24}$$

6.3 Attack detection

The final model of normal user behavior consists of minimal $x_{min,j}$ and maximal $x_{max,j}$ feature values, centroids $\mu_i = m(p_i)$, numbers of vectors assigned $w_i = w(P_i)$ and sums of squared feature values $\varsigma_i = \varsigma(P_i)$ for k connection partitions $p_1,\dots,p_k$ and centroids $M_i = m(P_i)$, numbers of vectors assigned $W_i = w(P_i)$ and matrices $S_i = S(P_i)$ for K session partitions $P_1, \dots, P_K$.

First, we recalculate conversation centroids and sums of squared feature values according to values $x_{min,j}$ and $x_{max,j}$ used for the standardization:

$$m_{ij} = \frac{\mu_{ij} - x_{min,j}}{x_{max,j} - x_{min,j}},$$
$$s_{ij} = \frac{\varsigma_{ij} + w_i(x_{min,j}^2 - 2x_{min,j}\mu_{ij})}{(x_{max,j} - x_{min,j})^2}, \tag{25}$$

where $i \in \{1, \dots, k\}$.

For each partition p_i we calculate radius r_i and diameter ψ_i in a similar manner they are calculated for cluster features in [26]:

$$r_i = \sqrt{\frac{e^T s_i}{w_i} - m_i^T m_i},$$
$$\psi_i = \sqrt{\frac{2w_i e^T s_i - 2w_i^2 m_i^T m_i}{w_i(w_i - 1)}}, \tag{26}$$

where e is vector of the same length as s_i each element of which is equal to 1.

Similarly, for each partition P_i we calculate radius R_i and diameter $\Psi_i$as follows:

$$R_i = \sqrt{\frac{E^T S_i^{diag}}{W_i} - M_i^T M_i},$$
$$\Psi_i = \sqrt{\frac{2W_i E^T S_i^{diag} - 2W_i^2 M_i^T M_i}{W_i(W_i - 1)}}, \tag{27}$$

where S_i^{diag} is the vector which consists of diagonal elements of matrix S_i and E is vector of the same length as S_i^{diag} each element of which is equal to 1.

If the distance d between standardized feature vector x extracted from a new connection and the closest centroid $m_{i(x)}$ is greater than the following linear combination of $r_{i(x)}$ and $\psi_{i(x)}$:

$$d(x, m_{i(x)}) > r_{i(x)} + \alpha\psi_{i(x)}, \tag{28}$$

then this connection is classified as an attack.

Similarly, if the distance d between feature vector y extracted from a new session and the closest centroid $M_{i(y)}$ is such that:

$$d(y, M_{i(y)}) > R_{i(y)} + \beta\Psi_{i(y)}, \tag{29}$$

then this user session is classified as an attack. Parameters $\alpha > 0$ and $\beta > 0$ are supposed to be tuned during the model validation stage in order to guarantee the highest detection accuracy.

7 Algorithm evaluation

In order to evaluate the detection approach proposed, first, we briefly overview our virtual network environment used to generate some realistic end user network traffic and various DDoS attacks. Then, we evaluate the cost of the clustering on the training data. Finally, we present results of the detection of three different DDoS attacks.

7.1 Test environment and data set

To test the DDoS detection scheme proposed in this study, a simple virtual network environment is designed. The environment botnet consists of a command and control (C2) center, a web server, and several virtual bots. C2 stores all necessary information about bots in a data base and allows to control bots by specifying the traffic type, the pause between two adjacent sessions and the delay between connections in one session. The web server has several services including a web bank website, file storage, video streaming service and few others. Each bot is a virtual machine with running a special program implemented in Java that allows the bot to receive commands from C2 and generate some traffic to the web server. It is worth noting that all the traffic is transferred by using encrypted SSL/TLS protocol.

In this research, we concentrate on the analysis of incoming and outgoing traffic of the bank web site that allows a client to log in and do some bank operations, see Figure 1. In order to generate a normal bank user traffic, we specify several scenarios that each bot follows when using the site. Each scenario consists of several actions following each other. These actions include logging in to the system by using the corresponding user account, checking the account balance, transferring some money to another account, checking the result of the transaction, logging out of the system, and some other actions. Each action corresponds to requesting a certain page of the bank service with all of its embedded content. Pauses between two adjacent actions are selected in a way similar to a human user behavior. For example, checking the account's balance usually takes only a couple of seconds, whereas filling in information to transfer money to another account may take much longer time.

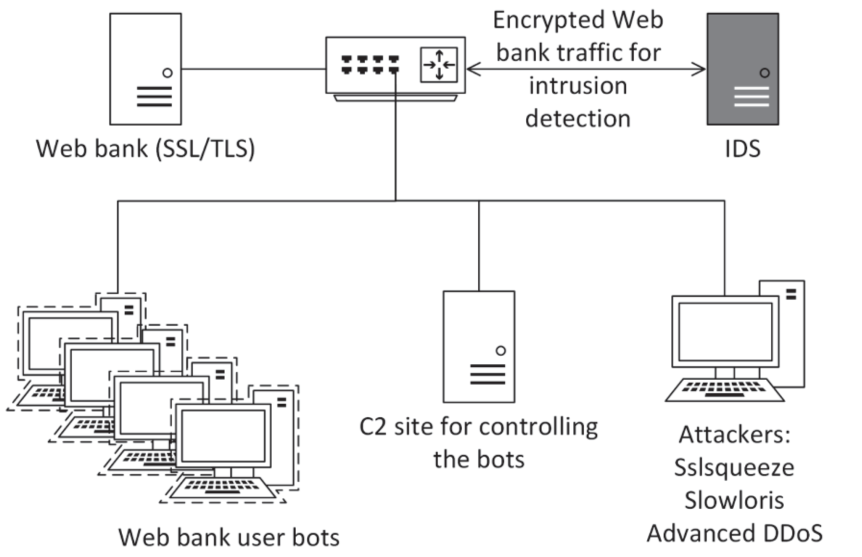
Encrypted Web bank traffic for intrusion detection
Web bank (SSL/TLS)
IDS
Web bank user bots
C2 site for controlling the bots
Attackers: Sslsqueeze Slowloris Advanced DDoS

Figure 1. Test setup

In addition to the normal traffic, several attacks are performed against the bank web service. First DDoS attack tested is Sslsqueeze. During this attack, attackers send some bogus data to the server instead of encrypted client key exchange messages. By the moment the server completes the decryption of the bogus data and understands that the data decrypted is not a valid key exchange message, the purpose of overloading the server with the cryptographically expensive decryption has been already achieved.

The second attack is Slowloris. In the case of this attack, each attacker tries to hold its connections with the server open as long as possible by periodically sending subsequent HTTP headers, adding to-but never completing-the requests. As a result, the web server keeps these connections open, filling its maximum concurrent connection pool, eventually denying additional connection attempts from clients. Moreover, we carry out a more advanced DDoS attack with the attackers that try to mimic the browsing behavior of a regular human user. During this attack several bots request sequences of web pages with all of their embedded content from the service. Unlike the normal user behavior, these sequences are not related to each other by any logic but generated randomly.

Finally, an intrusion detection system (IDS) prototype that relies on the proposed technique is implemented in Python. The resulting program analyzes arriving raw packets, combines them to conversations, extracts necessary features from them, implants the resulting feature vectors to the model in the training mode and classifies those vectors in the detection mode. The IDS is trained with the traffic that does not contain attacks by using the online training algorithm proposed. After that, the system tries to find conversations and clients that are related to the attacks specified above.

The resulting program analyses arriving raw packets, combines them to conversations, extracts necessary features from them, implants the resulting feature vectors to the model in the training mode (Figure 2) and classifies those vectors in the detection mode (Figure 3).

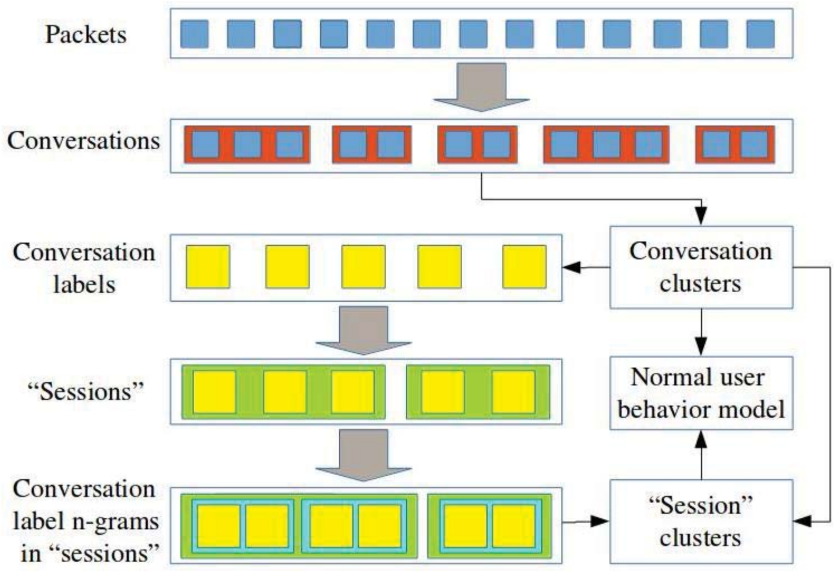

Figure 2. IDS in the training mode

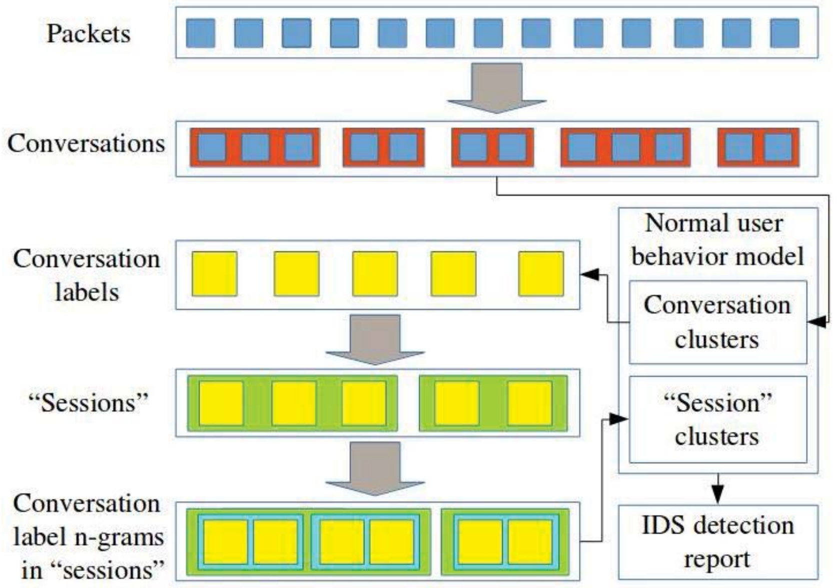

Figure 3. IDS in the detection mode

7.2 Clustering evaluation

In order to evaluate the clustering scheme, we generate a portion of normal user traffic as described above. The duration of the simulation is 10 minutes, during which 45 bots communicate with the web bank server. The pause between two adjacent sessions varies from 15 to 45 seconds. To train the system we use the time window of length 5 seconds. Cluster centroids in the model are compressed every 10 time windows.

In order to evaluate the clustering scheme, we calculate the average cost C of dividing the resulting set of vectors X into clusters with centers m as follows:

$$C = \frac{1}{|X|} \sum_{x \epsilon X} min_{y \epsilon m} d(x, y). \quad (30)$$

First, we compare the cost of clustering vectors that represent conversations by offline and online approaches. The comparison results for different numbers of clusters are presented in Figure 4. As one can see, the cost of clustering these vectors online is comparable with the cost in the offline case and the difference between costs reduces when the number of clusters grows. When the number of clusters is equal or greater than 15, the difference between costs is only about 1.4%.

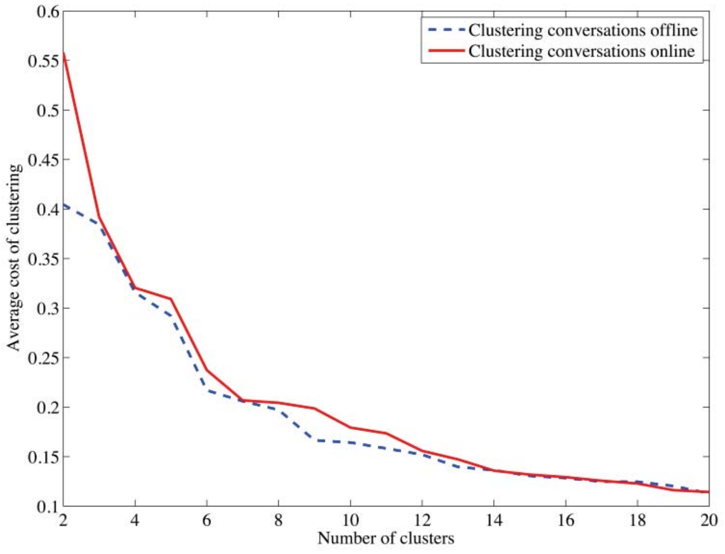

Figure 4. Average cost of clustering conversations

Similar results are obtained for n-gram vectors that represent normal user session approximations. It is worth noticing that, for session clusters built in the online mode, n-grams are constructed based on the clustering conversation vectors in online mode, and correspondingly, for session clusters obtained in offline mode, n-grams are constructed based on the clustering conversation vectors in offline mode. The comparison results for different numbers of clusters are presented in Figure 5. As one can notice, the cost of clustering in the online case is again comparable with the offline approach.

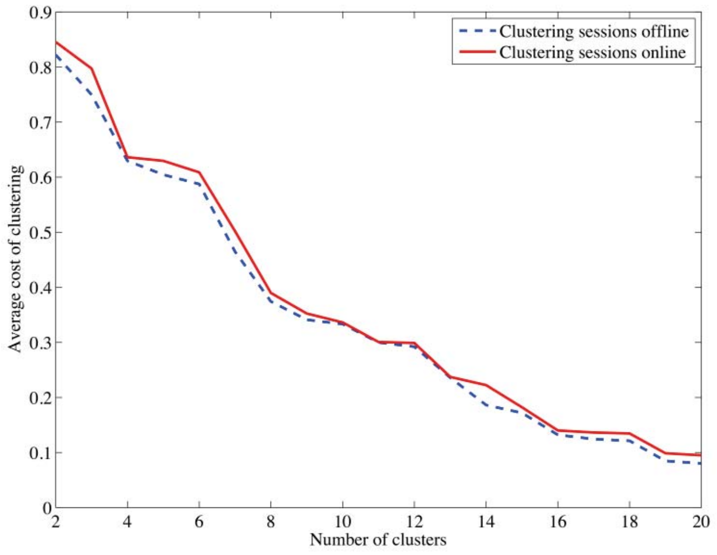

Figure 5. Average cost of clustering ”sessions”

7.3 Detection accuracy

In order to evaluate attack detection accuracy, we employ the training dataset described above. The IDS is trained with the traffic that does not contain attacks by using the training algorithm proposed. Once the training has been completed, several attacks are performed to evaluate true positive rate (TPR), false positive rate (FPR) and accuracy of the algorithms. Since one of the most important drawbacks of an anomaly-based detection system is high numbers of false alarms, we are only interested in results when the false positive rate is below 1%.

We expect that trivial DDoS attacks such as Sslsqueeze and Slowloris can be detected by analyzing feature vectors extracted from conversations. Since almost every conversation between a client and the web server takes few milliseconds, the time window size is selected to be 1 second. Since, we could not find any other anomaly-based attack detection method for the encrypted network traffic, we compare our results with the offline version of the algorithm proposed in this study. In [17], we compared several well-known batch clustering algorithms for the problem of trivial DDoS attack detection. It turned out, that such algorithms as k-means, self-organizing maps and fuzzy c-means allow one to detect slow HTTP attacks with high accuracy. In this study, we use k-means combined with k-means++ as the offline version of our approach.

Figure 6 shows how TPR depends on FPR when detecting Sslsqueeze for different numbers of conversation clusters in the model of normal user behavior and different values of parameter α. As one can notice, for low values of FPR both variants of the algorithm are able to detect almost all conversations related to the attack. In particular, we are able to detect 98.5% of intrusive conversations with no false alarms and 99.9% of such conversations with FPR equal to 0.8%.

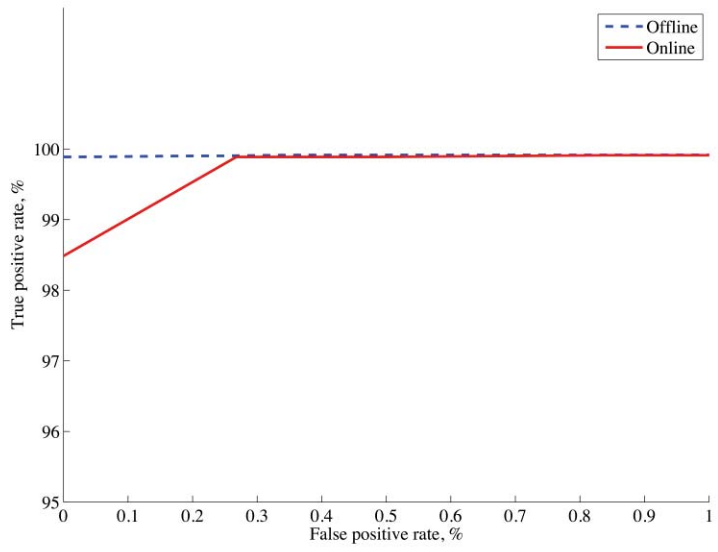

Figure 6. Dependence of true positive rate on false positive rate of Sslsqueeze detection for different clustering parameters

Dependency of TPR on FPR when detecting Slowloris is presented in Figure 7. As previously, different numbers of conversation clusters in the model of normal user behavior and different values of parameter α are used to obtain the results. As one can see, both variants of the algorithm are able to detect almost all conversations related to the attack (TPR > 99%) without false alarms.

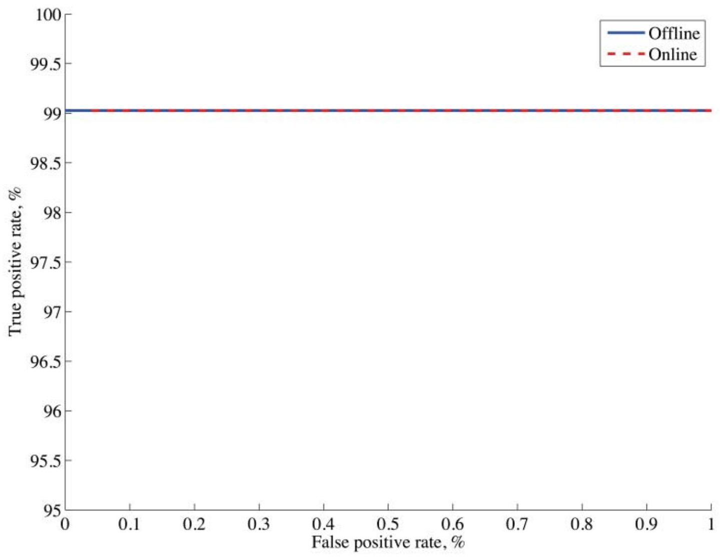

Figure 7. Dependence of true positive rate on false positive rate of Slowloris detection for different clustering parameters

Finally, we carry out a more advanced DDoS attack with the attackers that try to mimic the browsing behavior of a regular human user. During this attack several bots request a random sequence of web resources from the server. Since all those requests are legitimate, the corresponding conversations are classified as normal. However, the analysis of n-gram vectors that represent approximations of user sessions allow us to detect the most part of the attacking attempts. In this simulation, 2-gram model is applied. Figure 8 shows how TPR depends on FPR for different values of the time window size. When the time window duration is only 1 second, there is no possibility to detect the attack since the number of conversations in all user sessions is not enough to distinguish a legitimate user's sessions from attacker's ones. However, when the size of the time window grows, normal user sessions become more and more distinguishable from the sessions related to the attack. As one can see, when the time window size is 4 seconds, almost all intrusive sessions can be detected (TPR = 97%) without false alarms. When the time window size is 8 seconds, all sessions related to the attack can be detected (TPR = 100%) with few false alarms (FPR = 0.93%).

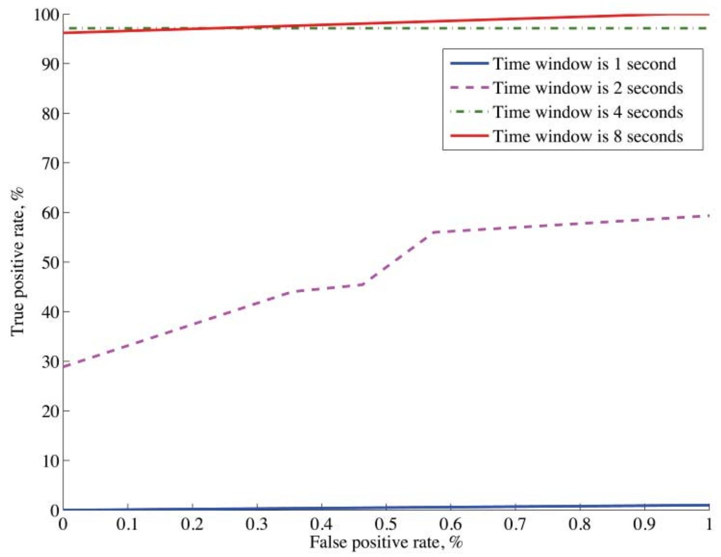

Figure 8. Dependence of true positive rate on false positive rate of more advanced DDoS attack detection for different time window sizes and clustering parameters

Table 1 shows the accuracy of detection of these three DDoS attacks for the cases when clustering parameters are selected in an optimal way, i.e. when the detection accuracy is maximal.

Table 1. Detection accuracy

Attack	Time window size			
	1 second	2 seconds	4 seconds	8 seconds
Sslsqueeze	99,89 %			
Slowloris	99,77 %			
Intermediate DDoS	63,29 %	85,21 %	98,52 %	99,34 %

7.4 Discussion

The detection approach proposed in this study allows us to detect trivial as well as more advanced DDoS attacks. However, the detection accuracy strongly depends on the clustering parameters selected for the model of normal user behavior. For this reason, these parameters should be carefully selected. For example, the number of conversation clusters should depend on the number of web pages of the web service considered. For completely different pages there should be different clusters whereas similar pages can be combined to the same cluster. Moreover, since clustering vectors that represent user sessions requires analysis of matrices of size $k^n \times k^n$, where k is the number of conversation clusters and n is the number of grams in the n-gram model, value of k should be limited depending on available memory. For the same reason, number of grams in the n-gram model should be small. As we can see from the results presented above, 2-gram model allows us to detect almost all intrusions. The number of ”session” clusters should depend on the number of different scenarios how the web service can be used by some user. Parameters α and β are recommended to lie between 0 and 1. If these parameters are close to zero, the number of true positives is the highest. When these parameters grow, the number of false alarms decreases. Finally, the size of time window should depend on the average duration of a normal user session in order to increase the accuracy of the detection of intermediate DDoS attacks.

It is also worth to notice, that despite our approach allows us to distinguish attacks from the normal user traffic with high accuracy, the traffic we label as normal would be classified as an advanced DDoS attack by [10], because this traffic consists of sequences of HTTPS requests which are carefully chosen so as to better mimic the browsing behavior of regular human users. For this reason, in the future, we are going to test our method on some real end user traffic.

8 Conclusion

In this research, we considered the problem of timely detection of different sorts of application-layer DDoS attacks in encrypted high-speed network traffic by applying an anomaly detection-based approach to statistics extracted from network packets. Our method relies on the construction of the normal user behavior model by applying a data stream clustering algorithm. The online training scheme proposed allows one to rebuild this model every time when a new portion of network traffic is available for the analysis. Moreover, it does not require a lot of computing and memory resources to be able to work even in the case of high-speed networks. In addition, an IDS prototype that relies on the proposed technique was implemented. This prototype was used to test our technique on the data obtained with the help of our virtual network environment that generated realistic traffic patterns of end users. As a result, almost all DDoS attacks were properly detected, while the number of false alarms remained very low.

In the future, we are planning to improve the algorithm in terms of the detection accuracy, and test it with a bigger dataset which contains real end user traffic captured during several days. In addition, we will focus on the simulation of more advanced DDoS attacks and detection of these attacks by applying our anomaly-based approach.

9 Acknowledgment

This work was partially funded by the Regional Council of Central Finland/Council of Tampere Region and European Regional Development Fund/Leverage from the EU 2014-2020 as part of the JYVSECTEC Center project of JAMK University of Applied Sciences Institute of Information Technology.

10 References

[1] Kaspersky Lab, “Statistics on botnet-assisted DDoS attacks in Q1 2015”, 2015, https://securelist.com/blog/research/70071/statistics-on-botnet-assisted-ddos-attacks-in-q1-2015/, accessed 26.8.2016.

[2] Radware, “2015-2016 Global Application & Network Security Report”, 2016, https://www.radware.com/ert-report-2015/?utm_source=security_radware&utm_medium=promo&utm_campaign=2015-ERT-Report, accessed 26.8.2016.

[3] Verisign, “Distributed Denial of Service Trends Report”, 2016, https://www.verisign.com/en_US/security-services/ddos-protection/ddos-report/index.xhtml, accessed 26.8.2016.

[4] V. Durcekova, L. Schwartz, and N. Shahmehri, “Sophisticated Denial of Service Attacks Aimed at Application Layer”, In Proceedings of the 9th International Conference ELEKTRO, pp 55–60, 2012.

[5] R. Chen, J. Wei and H. Yu, “An improved Grey Self-Organizing Map based DOS detection”, 2008 IEEE Conference on Cybernetics and Intelligent Systems, Chengdu, 2008, pp. 497-502.

[6] Y. Ke-xin and Z. Jian-qi, “A Novel DoS Detection Mechanism”, In Proceedings of International Conference on Mechatronic Science, Electric Engineering and Computer (MEC), pp 296–298, 2011.

[7] J. Yuan and K. Mills, “Monitoring the Macroscopic Effect of DDoS Flooding Attacks”, IEEE Tran. on Dependable and Secure Computing, vol. 2, pp 324–335, 2005.

[8] M. Aiello, E. Cambiaso, M. Mongelli, and G. Papaleo, “An On-Line Intrusion Detection Approach to Identify Low-Rate DoS Attacks”, In Proceedings of International Carnahan Conference on Security Technology (ICCST), pp 1–6, 2014.

[9] P. Chwalinski, R. R. Belavkin, and X. Cheng, “Detection of Application Layer DDoS Attacks with Clustering and Bayes Factors”, In Proceedings of IEEE International Conference on Systems, Man, and Cybernetics (SMC), pp 156–161, 2013.

[10] D. Stevanovic and N. Vlajic, “Next Generation Application-Layer DDoS Defences: Applying the Concepts of Outlier Detectionin Data Streams with Concept Drift”, In Proceedings of the 13th International Conference on Machine Learning and Applications, pp 456–462, 2014.

[11] C. Xu, G. Zhao, G. Xie, and S. Yu, “Detection on Application Layer DDoS using Random Walk Model”, In Proceedings of IEEE International Conference on Communications (ICC), pp 707–712, 2014.

[12] S. Yadav and S. Selvakumar, “Detection of Application Layer DDoS Attack by Modeling User Behavior Using Logistic Regression”, In Proceedings of the 4th International Conference on Reliability, Infocom Technologies and Optimization (ICRITO) (Trends and Future Directions), pp 1–6, 2015.

[13] S. Yadav and S. Selvakumar. “Detection of Application Layer DDoS Attack by Feature Learning Using Stacked Autoencoder”, In Proceedings of International Conference on Computational Techniques in Information and Communication Technologies (ICCTICT), pp 261–266, 2016.

[14] C. Ye, K. Zheng, and C. She, “Application layer DDoS detection using clustering analysis”, In Proceedings of the 2nd International Conference on Computer Science and Network Technology (ICCSNT), pp 1038–1041, 2012.

[15] J. Zhang, Z. Qin, L. Ou, P. Jiang, J. Liu, and A. Liu, “An Advanced Entropy-Based DDOS Detection Scheme”, In Proceedings of International Conference on Information Networking and Automation (ICINA), pp 67–71, 2010.

[16] M. Zolotukhin, T. Hämäläinen, T. Kokkonen, A. Niemelä, and J. Siltanen. Data mining approach for detection of ddos attacks utilizing SSL/TLS protocol. Springer Lecture Notes in Computer Science, vol. 9247, pp 274–285, 2015.

[17] M. Zolotukhin, T. Hämäläinen, T. Kokkonen, and J. Siltanen. Increasing web service availability by detecting application-layer ddos attacks in encrypted traffic. In Prodeedings of the 23rd International Conference on Telecommunications (ICT), pp 1–6, 2016.

[18] M. Trojnara, “Sslsqueeze 1.0”, SSL service load generator proof of concept, 2011, http://pastebin.com/AgLQzL6L, accessed 26.8.2016.

[19] Gartner, “Are Cybercriminals hiding in SSL traffic”, White paper sponsored by venafi, 2015, https://citrixready.citrix.com/content/dam/ready/partners/ve/venafi/venafi-trustforce/Gartner%20-%202015%20-%20Cybercriminals%20Hiding%20in%20SSL%20Traffic.pdf, accessed 26.8.2016.

[20] W. Bulajoul, A. James, and M. Pannu, “Network intrusion detection systems in high-speed traffic in computer networks”, In Proceedings of the 10th IEEE International Conference one-Business Engineering (ICEBE), pp 168–175, 2013.

[21] K. Li, W. Zhou, P. Li, J. Hai, and J. Liu, “Distinguishing DDoS Attacks from Flash Crowds Using Probability Metrics”, In Proceedings of the 3rd International Conference on Network and System Security (NSS), pp 9–17, 2009.

[22] U. Chaudhary, I. Papapanagiotou, and M. Devetsikiotis, “Flow Classification Using Clustering And Association Rule Mining”. In Proceedings of the 15th IEEE International Workshop on Computer Aided Modeling, Analysis and Design of Communication Links and Networks (CAMAD), pp 76–80, 2010.

[23] D. Arndt and A. N. Zincir-Heywood, “A Comparison of Three Machine Learning Techniques for Encrypted Network Traffic Analysis”, In Proceedings of IEEE Symposium on Computational Intelligence for Security and Defense Applications (CISDA), pp 107–114, 2011.

[24] R. Chitta, R. Jin, and A. Jain, “Stream Clustering: Efficient Kernel-based Approximation using Importance Sampling”, In Proceedings of IEEE International Conference on Data Mining Workshop (ICDMW), pp 607–614, 2015.

[25] J. A. Silva, E. R. Faria, R. C. Barros, E. R. Hruschka, A. C. de Carvalho, and J. Gama, “Data Stream Clustering: A Survey”, ACM Computing Surveys, vol. 46, article 13, pp 13:1–13:31, 2013.

[26] T. Zhang, R. Ramakrishnan, and M. Livny, “BIRCH: A New Data Clustering Algorithm and Its Applications”, Data Mining and Knowledge Discovery, vol. 1, no. 2, pp 141–182, 1997.

[27] C. Aggarwal, J. Han, J. Wang, and P. Yu, “A framework for clustering evolving data streams”, In Proceedings of Conference on Very Large Data Bases (VLDB03), vol. 29, pp 81–92, 2003.

[28] P. Kranen, I. Assent, C. Baldauf, and T. Seidl, “The ClusTree: indexing micro-clusters for anytime stream mining”, Knowl. Inf. Syst., vol. 29, issue 2, pp 249–272, 2011.

[29] M. Ackermann, M. Märtens, C. Raupach, K. Swierkot, C. Lammersen, and C. Sohler. “StreamKM++: A clustering algorithm for data streams”, ACM J. Exp. Algor. vol. 17, no. 2, article 2.4, 30 pages, May 2012.

[30] Y. Chen and L. Tu, “Density-Based Clustering for Real-Time Stream Data”, In Proceedings of the 13th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, pp 133–142, 2007.

[31] J. Gama, P. Rodrigues, and L. Lopes, “Clustering distributed sensor data streams using local processing and reduced communication”, Intell. Data Anal., vol. 15, issue 1 pp 3–28, 2011.

[32] M. Shindler, A. Wong, and A. W. Meyerson, “Fast and Accurate k-means For Large Datasets”, In Proceedings of the Conference on Neural Information Processing Systems, pp 2375–2383, 2011.

[33] T. Hirsimäki, J. Pylkkönen, and M. Kurimo, “Importance of High-Order N-Gram Models in Morph-Based Speech Recognition”, in IEEE Transactions on Audio, Speech, and Language Processing, vol. 17, no. 4, pp. 724-732, May 2009.

[34] C. Y. Suen, “n-Gram Statistics for Natural Language Understanding and Text Processing”, IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. PAMI-1, no. 2, pp. 164-172, April 1979.

[35] P. Domingos and G. Hulten. A general method for scaling up machine learning algorithms and its application to clustering. In Proceedings of the 8th International Conference on Machine Learning, pp 106–113, 2001.

[36] R. Shah, S. Krishnaswamy, and M. Gaber. Resource-aware very fast k-means for ubiquitous data stream mining. In Proceedings of the 2nd International Workshop on Knowledge Discovery in Data Streams, Held in Conjunction with the 16th European Conference on Machine Learning, 2005.

[37] D. Arthur and S. Vassilvitskii, K-means++: The Advantages of Careful Seeding, Proceedings of the Eighteenth Annual ACM-SIAM Symposium on Discrete Algorithms. pp 1027-1035, 2007.

[38] V. Braverman, A. Meyerson, R. Ostrovsky, A. Roytman, M. Shindler, and B. Tagiku. Streaming k-means on well-clusterable data. In Proceedings of the 22nd Annual ACM-SIAM Symposium on Discrete Algorithms, pp 26–40, 2011.