Stealthy Denial of Service (DoS) attack modelling and detection for HTTP/2 services

Author:
Adi, E; Baig, Z; Hingston, P

Publication details:
Journal of Network and Computer Applications
v. 91
pp. 1 - 13
1084-8045 (ISSN); 1095-8592 (ISSN)

Publication Date:
2017-08-01

Publisher DOI:
https://doi.org/10.1016/j.jnca.2017.04.015

Stealthy Denial of Service (DoS) Attack Modelling and Detection for HTTP/2 Services

Erwin Adi

Zubair Baig*

Philip Hingston

Edith Cowan University

https://doi.org/10.1016/j.jnca.2017.04.015

Abstract

A malicious attack that can prevent establishment of Internet connections to web servers is termed as a Denial of Service (DoS) attack; volume and intensity of which is rapidly growing thanks to the readily available attack tools and the ever-increasing network bandwidths. Contemporary web servers are increasingly vulnerable to such attacks. With the emergence of HTTP/2 as the successor of HTTP/1.x, existing techniques for detecting DoS attacks will not be entirely effective. Though nearly 90% of all contemporary web servers as yet have not migrated to HTTP/2, DoS attack modelling and detection is essential to prevent impending attacks of such kind from the adversary class. This study presents a model of DoS attack traffic that can be directed towards HTTP/2 web servers. The research conducted also extends previous studies that provided DoS attack models against HTTP/2 services, to present a novel and stealthy DoS attack variant that can disrupt routine web services, covertly. The attack traffic analysis conducted in this study employed four machine learning techniques, namely Naïve Bayes, Decision Tree, JRip and Support Vector Machines, and stealthy traffic properties are shown through having higher percentages of False

*Corresponding author
Email addresses: z.baig@ecu.edu.au (Zubair Baig)

Alarms. Results obtained through simulation show promise, and arguments are put forth on how future work can extend the proposed model to create further attack traffic models that may cause severe web service disruptions.

Key words: HTTP/2, Denial of Service attack, Traffic analysis, Machine learning techniques

1. Introduction

Businesses and modern society rely on Internet services for their communication and information needs. Ensuring the availability of these services is a challenging task due to the growing volume of Internet traffic and the various communication standards that it supports. The current web browsing standard (HTTP/1.1) is reaching its full capacity [1], as it was designed to exchange text. Web users often experience slow Internet speed; hence, a new standard (HTTP/2) has been designed to support communication at higher speed [2]. While the web programming software used to build websites remains unchanged, the HTTP/2 data communication techniques differ from those of HTTP/1.1. HTTP/2 architecture introduces binary framing, multiplexing, message interleaving, and application-layer flow control [2]. As such, the traffic it sustains over the Internet media shows different patterns than what were previously observed and reported in the literature, for HTTP/1.1. Through this contribution, novel techniques are proposed and evaluated for detecting malicious network traffic that target HTTP/2 services, as part of a Denial of Service (DoS) attack.

For monetary gain, the adversarial threat against the availability of businesses comprises attempts to bring down web servers and corresponding services [3, 4]. Such a threat can translate into a Denial of Service (DoS) attack, which is defined as an explicit attempt of an attacker to prevent legitimate users of a service from using the service [5]. Other motivations for launching DoS attacks include ideological beliefs to uphold one's views while attempting to suppress the opposition's ability to publicise through websites; accepting an intellectual challenge to learn how to launch attacks; and cyber warfare, i.e. attacks supported

by military or terrorist organizations [6]. Detecting and bearing the ability to prevent DoS attacks against a web server are therefore crucial in order to provide uninterrupted services to legitimate clients. For network and web hosting operators, the ability to detect DoS traffic prevents unwanted operational costs, helps to plan future infrastructure, and allows operators to provide services that otherwise would have been disrupted by illegitimate traffic.

In recent times, DoS attacks have been increasing in their volume, ubiquity, complexity, and use of novel techniques [4]. The attacks can be amplified using a network of compromised computers (*aka* botnet) in order to launch a storm of network traffic [7]. This variant is called a Distributed Denial-of-Service (DDoS) attack. In fact, flooding attacks reported after 1999 have been mostly DDoS in nature [6]. In 2002, the total volume of DDoS attack traffic against large carriers and content providers around the world was found to be 400 Mbps [8]. In 2013, this attack traffic volume was increased to 60 Gbps [9]. Recently the total worldwide attack traffic volume has been increasingly difficult to collect as the number has grown exponentially. To illustrate, in the second quarter of 2015 alone, DDoS attack volume touched a staggering 1,000 Gbps [10]. In the beginning of 2016, the British Broadcasting Corporation (BBC) website was flooded with a high intensity attack that produced attack traffic at 600 Gbps [11]. The threat thus posed to contemporary web servers cannot be underestimated, and the best approach towards securing web servers from such attacks is to have security controls such as intrusion detection systems, perform attack detection with a high degree of accuracy.

Current detection techniques for DoS attacks against web services are based on HTTP/1.1 traffic patterns. HTTP/2 opens up a new channel of opportunity for the adversary class to interrupt the availability of web servers. Analysis of HTTP/2 legitimate traffic and modeling and detection of DoS attack traffic against HTTP/2, are essential. Previous studies show that a flood of HTTP/2 packets can be modelled to bypass a hypothetical intrusion-detection system that monitors a computing resource's memory consumption [12] and CPU usage [13]. In this paper, we propose a model of DoS attack traffic i.e. stealthy

attacks, that executes the attack vector through intelligent crafting of network traffic intensities spread over a longer period of time. Consequently, the attack remains largely undetected by an intrusion detection system. We rank those network traffic features that are most relevant for improving the accuracy of the intelligent intrusion detection systems. The machine learning techniques were further evaluated on their ability to distinguish legitimate from attack traffic.

The contributions of this paper can be summarized as follows:

- Modeling of legitimate network traffic,
- Modeling of two stealthy DoS attack variants,
- Proposal of an attack detection scheme,
- Simulation of the proposed scheme and analysis of obtained results, and
- Proposal of future directions of research.

2. Background

Hypertext Transfer Protocol (HTTP) has been the protocol of choice for web browsing communication. The current version of the protocol, HTTP/1.1, was designed to transfer text over the Internet (TCP/IP based). As technology evolved, rich media was being increasingly transferred using the same protocol, causing the web response time to increase. Furthermore, modern web applications that use these rich media have created a demand for more user interactions, causing the protocol to reach its limit. Consequently, web users experience slow connections to websites.

The HTTP/2 protocol format is based on binary framing as opposed to the newline-delimited plain text mechanism of its predecessor. Binary framing allows its parser to efficiently identify the location of the subsequent packets in the traffic flow, and quickly identify each packet's type and flags. The binary framing also allows multiplexing, i.e., multiple requests/responses in one TCP

connection per origin. To implement multiplexing, the protocol tags each packet with a stream ID. Packets with different stream IDs can be communicated independently, allowing virtual communication streams in one TCP connection.

HTTP/2 introduces a *flow control* mechanism in order for the communicating devices to advertise their susceptibility to congestion. The `window_update` frame is an HTTP/2 message that implements flow control. The purpose of flow control is to moderate the many streams established between a client and a server, in a single connection. This allows a server to balance packet flows between the streams of a single connection. The window_update frame is one such frame that allows a client or a server to communicate the size of data that the sender can transmit in a single frame, in addition to the current size. These capabilities did not exist in HTTP/1.1.

Several techniques have been reported in the literature to identify flood-based attacks including those using statistics, entropy, and wavelet analysis. In addition, machine learning techniques are also popular for traffic analysis, since they are able to classify large data sets that exist in multi-dimensional spaces. The machine learning techniques employed in this study for legitimate and attack traffic differentiation are Naïve Bayes (NB), Decision Tree (DT), JRip, and Support Vector Machines (SVMs).

Naïve Bayes (NB) is one of the most widely used techniques in data mining communities [14]. It is used effectively in many studies on traffic analysis and DoS detection. For instance, a study applied Naïve Bayes to classify traffic without inspecting the payload of the traffic [15]. The dataset was obtained by extracting features from the TCP headers of the observed traffic. The study also showed that it achieved an accuracy of 95% in classifying the traffic through application of Naïve Bayes.

One of the biggest challenges in classifying network traffic is the large volume of data to analyse, given a set of features. One study proposed a feature ranking technique using Naïve Bayes [16], thus reducing the dimensionality of the data for faster computation. To compare the performance of the proposed technique, other widely used feature ranking techniques, such as Information Gain and

Gain Ratio, were also applied by the authors. The proposed technique in the study showed that it outperformed the widely used feature ranking techniques when applied on large volumes of data.

In contrast, another study proposed a solution when too few data samples were available [17]. The study proposed a technique to pre-process traffic before extracting features to be classified using Naïve Bayes. The technique correlated traffic flows that were generated from the same application. The study showed that the proposed method outperformed other machine learning techniques such as Decision Tree and k-NN.

Decision Tree (DT) is one of the most popular techniques applied for data classification [14]. It is a sequence of rules wherein the current selected rule decides the subsequent rules to be selected by splitting the rule into two or more branches forming a tree-like structure. Decision Trees have also been applied in traffic analysis research for identifying botnets [18] and network anomalies [19]. Botnets are not only deployed to generate DDoS traffic, but also help spread spam mails and to steal passwords. A study reported use of Decision Tree to identify botnet behaviour from generated traffic patterns [18]. The scheme compared its performance analysis with Naïve Bayes and concluded that Decision Trees can produce better classification accuracies. Interestingly, higher detection rates were achieved when analysis was done on HTTP-filtered traffic, wherein only HTTP traffic was introduced to the classifier. The study suggested that the identified bots communicated using the HTTP/1.x protocol.

JRip is considered a faster machine learning technique than Decision Trees [20, 21]. It is based on rule-learning techniques, which classify data samples into a single class and seek a set of rules to best classify data. An extension of the technique named RIPPER introduced a pruning method that reduces the complexity of a tree [20]. JRip is a Java-based implementation of RIPPER. It was applied for traffic analysis studies to reduce false alarms [22], to select best traffic features [23] and to efficiently reduce the volume of data introduced to an intrusion detection system for classification [24].

Support Vector Machines (SVMs) [25] are an example of a supervised learn-

ing technique. They extend linear regression models to separate datasets whose classes are otherwise linearly inseparable. SVMs have been applied to classify DoS traffic and legitimate traffic in a recent study [26]. The study aimed to detect DoS in mobile Ad-hoc networks, which is a network of mobile devices that are connected wirelessly, wherein each device acts as a router in addition to its normal intended use. Such kinds of networks are very vulnerable to DoS attacks since there is no security policy imposed on access to these connected devices. The study showed that SVMs classify with greater than 90% accuracy in all conducted experiments.

SVMs can also separate multiclass data. For instance, a study reported in [27] aimed to classify 7 application types of traffic (bulk, interactive, mail, service, www, p2p, and 'others'). Prior to this study, traffic classification focused only on HTTP data. Results showed a 96.9% accuracy in classifying the conglomerate (legitimate and attack) traffic. The study also showed that the same method was able to classify homogeneous traffic comprising 87% HTTP traffic with 99.4% accuracy. This suggests that the method was externally valid, i.e. it was able to achieve a desired accuracy level when applied to diverse sets of data.

These machine learning techniques were used to analyse and detect DoS attack traffic where devices communicate using HTTP/1.x. In contrast, this study used machine learning techniques to analyse HTTP/2 attack and normal traffic. Two studies have been identified for HTTP/2 attack modelling and detection analysis. First, a study employed HTTP/2 window_update packets to model flooding-based attack against a target machine running HTTP/2 services [12]. This is in contrast to employing HTTP Request packets in traditional DoS attacks where devices communicate using HTTP/1.x. The study showed that a flood of HTTP/2 Request packets did not successfully incapacitate the target machine, while a flood of HTTP/2 window_update packets was able to incapacitate approximately 12 machines. Furthermore, the memory of the target machines was only consumed up to 2 MB. This suggests that the model was able to bypass a hypothetical intrusion-detection systems that monitored memory

consumption of a target machine.

The second study modelled HTTP/2 DDoS attacks that can bypass hypothetical intrusion-detection systems that monitor CPU usage of a target machine [13]. A flood was modelled as 131,000 window_update packets sent in 38.5 seconds, with its payload window-size-increment set to 16,384. The payload window-size-increment signified the additional HTTP/2 frame size in bytes that an attacking machine can send, in addition to its previous value. The study showed that an attacking machine sending such a flood caused a target machine to consume 50% of CPU usage. Because any normal computing activities, such as disk writing, can cause such indication, the traffic produced by the model can bypass hypothetical intrusion-detection systems that monitored CPU consumption of a target machine. The study showed that 4 attacking clients, where each client run 2 processes of traffic generation from the model, caused a target machine to continually show 100% CPU consumption.

3. Legitimate and Attack Traffic Models

3.1. Legitimate Traffic Model

While HTTP/2-enabled services are gaining popularity, currently most web servers still communicate using the HTTP/1.1 protocol. This implies that a sensor placed at a backbone of a computer network would not be able to tap much data on HTTP/2 traffic. Through this study, HTTP/2 traffic was subsequently modelled to mimic real user traffic and was generated as part of the experiments.

This section explains how legitimate user traffic can be modelled, and how HTTP/2 traffic can be generated from the defined model. Legitimate traffic was subsequently built-upon to create flash-crowd traffic, i.e. a large volume of legitimate traffic that incapacitates a web server, similar in fashion to a DoS attack, albeit with non-malicious intent. Figure 1 illustrates the framework of generating flash-crowd traffic from a model of normal user behaviours. The

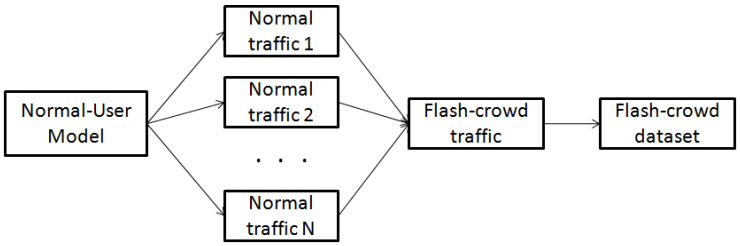


Figure 1: The framework of creating flash-crowd dataset from legitimate traffic model

proposed user behaviours adopted a publicly-available log that described user actions in the Internet.

3.1.1. Logs of Online User Browsing Behaviors

Behaviour of a normal Internet user was modelled from DOBBS, a publicly available log of user actions obtained from online browsing [28]. This log was comprised of records of user actions when a user is online, such as opening a new browser, adding a new browser tab, clicking a link or typing a web address, etc. The dataset was for a year-long activity record, and was collected from volunteers from around the world who installed a browser plugin that sent logs of these actions to a central archiving system.

There are three tables in the DOBBS log representing three types of events: first, that logged the browser's window-related data such as window minimized/maximized, browser tab opened/closed, and whether window is focused; second, that logged session-related data such as user inactive or idle (for example due to reading the information on the browser); and third, that was related to user actions as a result of browsing activities. Each table has a *user ID* column to identify the unique user who performed the actions. For the purpose of this study, the three tables were combined and the data was sorted based on the recorded time-stamp per user ID. This log showed a story-like event illustrating how a user browsed websites during a given period of time. An example of a data sample is shown in Table 1.

The table illustrates that a user initialised its browser, opened two tabs,

Table 1: A snippet of DOBBS Sample

Time	User ID	Event ID	Event description
20130826181127.900	48115555	100	New browser window opened
20130826181127.900	48115555	200	Session started
20130826181128.400	48115555	110	New browser tab opened
20130826181128.400	48115555	110	New browser tab opened
20130826181143.600	48115555	400	New web page loaded

and initiated web browsing after 15 seconds. The time-stamp for each entry presented how much time a given event took. In this study, a 3-day DOBBS data collection between 6 to 8 August 2013 was sampled, which is named DOBBS Sample. This period has the most number of users and surf entries. The DOBBS Sample was used to construct a User Model, i.e. a model that mimic normal user behaviours when online.

3.1.2. User Model

The User Model was represented in terms of states and transitions. Each recorded event was represented as a state with its own *dwell time* in that state, which was the time an event remained in one state before moving to another state. Each state led to one other state or more, and the probability of a state transitioning to another state was calculated by counting its frequency of occurrence in the actual DOBBS log. This effectively modelled one sample user that browsed web sites. Figure 2 shows the model of traffic that was described in Table 1. In the figure, there was an equal chance of state 110 to transit to either state 400 or to itself, because the frequency of those transitions in the data log was equal. As for the model used in this study, the transition probability of a state was tallied from the 3-day sample, i.e. the DOBBS Sample, which accommodates larger data than the example given in Table 1.

The User Model was constructed from a sequence of DOBBS Sample entries. As illustrated in Figure 3, the model had 16 states with many edges (transi-

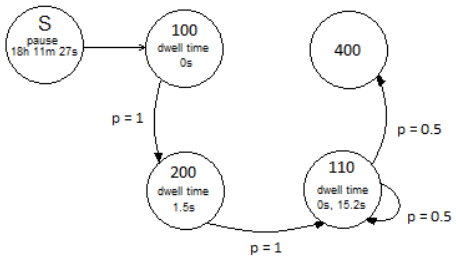


Figure 2: State Transition representing a User Model

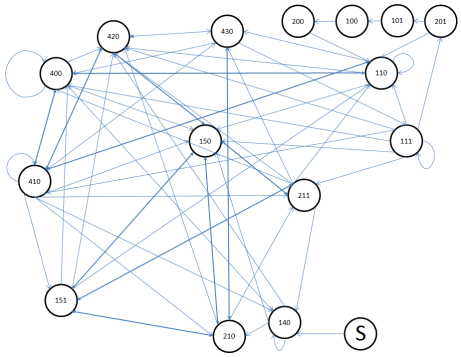


Figure 3: An example of one user model taken from DOBBS Sample

tions). The "S" state identified the first event observed in the log; therefore, it pointed to only one state in the model and acted as the starting state. The data structure of the model was coded using a two-dimensional matrix with I rows representing the current state, and J columns representing the next state. The data structure of each cell c_{ij} in the matrix had the dwell-time value for state i and the probability value to transit to the next state j.

The defined User Model was used to generate legitimate traffic.

3.1.3. Legitimate Traffic Model

This study was to generate a large volume of legitimate traffic that incapacitates a server, known as flash-crowd traffic. Two scenarios were implemented to generate a large volume of legitimate traffic. The first one was called *Ubot*, with the "U" standing for "User". It was defined to replay one User Model and generate legitimate traffic that mimicked the user. Ubot takes a User Model

as input, looks for a starting state and transits to other states after a given dwell-time. When Ubot is in a state that represents a user requesting a web page, Ubot generates an HTTP/2 Request packet. Since DOBBS Sample has 21 distinct user IDs in the log, this study was equipped with 21 unique user models depicting individual web-surfing behaviour. Ubot was run to simulate any one of these users and generate traffic according to the individual patterns.

The second scenario was named *BotMaster*, defined to run a large number of Ubot modules to generate flash-crowd traffic. Each Ubot module simulates a user. A number of Ubot modules running in tandem simulate different user behaviours, generating HTTP/2 traffic patterns that mimic normal users. While BotMaster was designed to run any number of Ubots, 200 was the optimum number to run on each of the virtual machines used in this study. It was observed that each virtual machine can run 200 Ubots without showing a race condition, i.e. a situation where the behaviour of one process affected another.

To generate a large traffic volume, the number of virtual machines, where each machine run a BotMaster, was incrementally added to the client-server system until the server showed signs of resource consumption. The study showed that the server reached 100% CPU consumption continually when 26 virtual machines were actively generating traffic directed towards the server. This represented $26 \times 200 = 5,200$ normal users visiting a website. The captured traffic at the server side represented flash-crowd traffic, because it was generated from legitimate traffic pattern and it consumed the CPU utilisation of the server.

The generated traffic was captured to create a legitimate dataset. A network monitoring tool TShark, which is a command-line version of Wireshark [29] was used to capture the flash-crowd traffic at the server side. The file format was named *packet-capture* (pcap); the traffic was captured over 8706 seconds, which occupied a pcap file of size 2 GB. Currently this number is the maximum pcap-format file size, when captured using TShark. The traffic captured by TShark was filtered twice, through the *direction filter* and the *protocol filter*. The first filter was to retain traffic only from client-to-server. This filtering procedure ascertained that the packet flow from client to server alone was captured, to

represent a DoS attack. The second filter was a protocol filter, which operated so that only traffic involved in an end-to-end communications was considered. Messages irrelevant to a remote HTTP/2 server, such as DNS, DHCP, and ARP messages, were not considered for creating the dataset in this study.

After the traffic was filtered with the direction and protocol filters, it was further characterised using the feature extraction procedure. A 3,600-second flash-crowd traffic was sampled for this purpose. That is, the feature extraction procedure extracted the 42 feature values of each time frame of length 1 second of flash-crowd traffic. A 1-second time slice was chosen instead of other arbitrary interval values (e.g. 2 seconds, 5 seconds), to convenience the data validation process during the simulations. For example, a 3,600-second captured traffic should produce a 3,600-row dataset. The result could be arranged in a table with its columns representing the feature values, and each of its rows representing 1-second traffic instances. This $42 \times 3,600$ table represents the HTTP/2 legitimate dataset. The legitimate dataset feature values can be used to analyse and detect other traffic types, such as attacks. Furthermore, the legitimate dataset allows this study to design stealthy attack traffic models, where some of their features values overlap with those of legitimate traffic.

3.2. Stealthy Attack Model

This study proposes stealthy attack models. Stealthy traffic is defined as traffic that yield higher percentages of False Alarms than the DDoS traffic proposed in a previous study [13], when analysed using machine learning techniques. It has been shown that the HTTP/2 DDoS attack traffic was able to bypass intrusion-detection systems that monitor CPU usage of a target machine, as the attack traffic generated by each attacking client only consumed 50% CPU usage of the target machine. However, in this study, we show that the DDoS traffic can be distinguished through comparing one of its feature values to that of legitimate traffic. For example, a feature that counts the number of packets carrying SYN flags per 1-second traffic instance, hereby named the 'count_syn' feature, is distinguishable. This is illustrated in Figure 4.

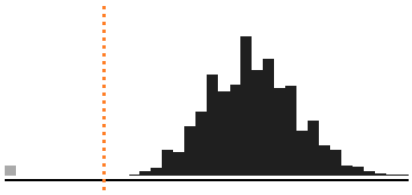


Figure 4: A threshold line can split DDoS and legitimate traffic

In Figure 4, the X-axis shows the number of SYN packets/sec, and the Y-axis shows the number of instances, or the tally for the X values. The black graph shows the distribution of the legitimate count_syn feature values, and the grey one shows that of DDoS traffic. It is unambiguous to choose a threshold value such as a vertical dotted-line to split the two colours as illustrated in the Figure, signifying that the DDoS traffic is distinguishable from legitimate traffic.

This study aimed to model attacks whose traffic continually consumed the victim's computing resource, yet caused machine learning techniques to incorrectly classify some traffic instances thereby yielding false alarms. In addition, the study tried to find the least possible number of attacking clients to successfully incapacitate a victim machine. In this study, attacking clients are named as 'bots'.

The study proposed to *camouflage* attack traffic with features of legitimate traffic. The stealthy attack model presented herewith comprises two groups of bots to camouflage attacks. One group attempts to exactly mimic the flash-crowd traffic features, and another does the generation of the offending traffic derived from the current understanding [13], i.e.: a flood of 131,000 window_update packets with window-size-increment payload set to 16,384 sent by an attacking bot for 38.5 ms caused a target server to consume 50% CPU consumption; and 4 attacking bots, where each bot sent such a flood caused a target server to consume 100% CPU resources. The mimicking bots are labelled as the *mime group*, and the attacking ones are labelled as the *offending group*.

The study controlled the amount of traffic generated by each group, until instances of 100% CPU consumption were observed at the victim machine.

To control the amount of traffic, two independent variables were added to the packet generator that constructed both the mime and offending bot anatomy. First, instead of indefinitely transmitting a flood of window_update packets as attempted in the previous sections, the bot sent intermittent floods. A flood was defined as 131,000 window_update packets in 38.5 ms. Here, *stealthy factor* is defined as the outcome of rolling an x-sided dice, where a random integer was generated between 1 and x. A flood from a bot towards a target server was launched when x equalled to 1. When $x \neq 1$, the bot sent only 1 HTTP/2 Request and then disconnected the TCP connection. Higher stealthy factor numbers imply smaller chances to have an outcome value of 1 out of a given stealthy factor x; hence, the higher the stealthy factor the less frequently for launching a flood. This mechanism created intermittent floods from the bots towards the victim rather than continuous attack traffic.

Second, a *delay* variable was introduced. Instead of pausing for 100 ms between streams as previously proposed [13], the bots disconnected the TCP connection and reconnected after a given delay. This variable controlled the flow of SYN packets/sec from each bot towards the victim. A SYN packet initiates a client-server TCP connection. Hence, the bots created SYN packets with a fixed delay between connections. The investigation in this section searched for a delay value between connections initiated by the mime group and the offending group. Larger delay values aided the mime group to mimic the number of SYN packets/second of flash-crowd traffic. However, larger delay values caused the offending group to send less attack traffic flow, causing the CPU consumption of the victim to slide from 100%.

In bot-induced DDoS attacks, the higher the number of bots, the closer the traffic pattern that they generated is to flash-crowd traffic [30]. This is reasonable, since both flash-crowd as well as attack traffic floods are generated from Internet-connected machines. In this study, a minimum number of bots were observed when the traffic that the bots generated caused a target machine to continually show 100% CPU consumption. An attacking bot anatomy is modelled as being built upon five parameters:

Table 2: Stealthy Attack-1 model

	Bot 1	Bot 2
Number of threads	1	1
Number of window_update	131K	131K
Stealthy factor	50	500
Delay between connections	11 ms	11 ms

- *number of threads*: the number of simultaneous processes runs on a bot machine, where each process can independently initiate a TCP connection with a remote machine, i.e. a target server.
- *number of window_update*: the number of window_update packets in each stream.
- *stealthy factor*: the frequency at which a TCP connection is used to send a flood of packets against a target machine, equals to $1/stealthyfactor$.
- *delay*: a time delay between successive TCP connections.

Two implementations of the model were proposed. The first model, named Stealthy Attack-1 (SA-1), is shown in Table 2. The table shows that the proposed stealthy attack traffic could be generated by simply two bots, with one bot representing the mime group and the other the offending group. One virtual machine was used to run each bot. Bot 1 acts as the offending group that sends 131K window_update packets as attack traffic towards the victim. The attack traffic is sent periodically with a stealthy factor equals to 50. This means the attack traffic is sent when a random variable x yields 1 of 50 chances, otherwise the bot sent 1 HTTP/2 Request and disconnected its TCP connection with the victim. Bot 2 sends less frequent attack traffic, as it is assigned a stealthy factor 500. This number is ten times higher than Bot 1, to maintain the desired number of TCP connections in its attempt to mimic flash-crowd traffic.

The second implementation of the stealthy attack model, named Stealthy Attack-2 (SA-2), is shown in Table 3. The model aimed to have attack traf-

Table 3: Stealthy Attack-2 model

	Bot 1	Bot 2	Bot 3	Bot 4
Number of threads	1	1	2	40
Number of window_update	131K	131K	0	0
Stealthy factor	5	5	n.a.	n.a.
Delay between connections	1 sec	1 sec	0.001 ms	5 sec

fic mimics another feature value of flash-crowd traffic, i.e. the size of packet carrying the RSK-ACK flag (henceforth named the 'size_rstAck' feature). This feature represents the total size of TCP packets with RST and ACK flags set observed in a 1-second traffic instance on the victim machine. For connection termination, a TCP packet with the RST flag set is sent by one bot machine to a remote machine. RST-ACK packets sent by the bot are essentially RST packets with the ACK flag set, to also acknowledge a previous packet received by the same machine. To further mimic the flash-crowd traffic, the Stealthy Attack-2 study proposed to have the mime-bot group closely mimic the values of the size_rstAck feature.

The intuition behind mimicking the size_rstAck values was due to varying implementations of the attack and the legitimate traffic. The former was implemented using `nghttp2` library [31], while the latter was using `curl` [32]. TCP connections implemented with nghttp2 library closes connections with RST packets, while connections curl library closes TCP connections with RST-ACK packets. It is acceptable to have various implementations of a communication standard, since standards also clearly indicate the implementation requirement levels that ranged from "must" to "optional", according to RFC 2119 [33]. Consequently different HTTP/2 libraries, while maintaining the requirements mandated by the HTTP/2 standard [2], are not uniform in their implementations, and subsequently the traffic pattern they produce varies.

The Stealthy Attack-2 model uses curl to implement the mime-bot group, while maintaining the use of nghttp2 as the engine for the offending group. It

was found that 4 bots were sufficient to cause a target server to continually show 100% CPU usage. Bots 1 and 2 are the offending group, launching a flood of attacking traffic periodically towards a target machine. The stealthy factor 5 meant that a flood was sent once every 5 seconds on average by chance. Hence, there was $1/5 \times 1/5 = 1/25$ chance during each second that the two bots simultaneously launched a flood of traffic towards a victim. Bots 3 and 4 formed the mime group that attempted to mimic the flash-crowd traffic. Because these two bots did not send any window_update traffic to the victim, any number assigned to the stealthy factor did not serve any purpose for launching attack packets. Hence, Table 3 shows "n.a." for the stealthy factors. The number of threads indicates the number of instances of the above scenario that were run by each bot during a given time frame. These threads, although relatively small in their number (2 for Bot 3, and 40 for Bot 4), were an attempt to mimic the 5,200 users that comprised flash-crowd traffic.

4. Attack Detection Scheme

In this section, we present the phases of operation of the attack traffic classification scheme. Figure 5 describes the process for classifying attack and legitimate traffic. Initially, a set of features from both the attack and legitimate traffic were extracted. The feature extraction yielded a dataset with 549 attack traffic instances for Stealthy Attack-1, and 254 instances for Stealthy Attack-2. The dataset was then merged with the legitimate dataset which consisted of 3,600 instances, yielding two datasets, one with a total of 4,149 instances and another with 3,854 instances. The features from each dataset were ranked using two features selection techniques, Information Gain and Gain Ratio. The study applied four machine learning techniques, i.e. Naïve Bayes (NB), Decision Tree J48 (DT), JRip and Support Vector Machines (SVMs), to classify the attack traffic when subjected to the legitimate traffic.

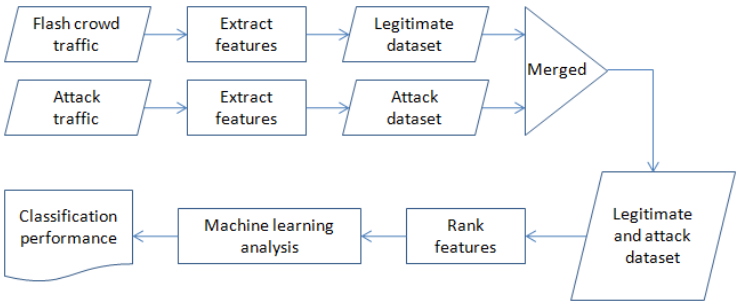


Figure 5: Process for legitimate and attack traffic classification

4.1. Feature Extraction

Upon introduction of both attack and legitimate network traffic, features were extracted and datasets were generated to represent legitimate and malicious traffic. The study observed patterns from the generated traffic, such as the packet types and their statistical properties such as the count, size, and lapse time of each packet since its connection initiation. For each 1-second traffic instance, the values of these features were obtained as follows.

- The *count* feature is the number of packets captured, grouped by packet type.
- The *size* feature is the total number of bytes of a packet captured, grouped by packet type.
- The *lapse* feature is the time lapse between packet capture and connection initiation (i.e. the length of time between a packet and the SYN packet of a connection), grouped by packet type. For each packet type:
  - If there is more than one packet within a connection, only the lapse value of the first packet is considered.
  - Because there can be more than one connection within an instance, there are as many lapse values as the number of connections. The lapse feature considers the *minimum*, *average*, as well as *maximum* values.

The values of count and size features characterize the volume of traffic being analysed. The values of lapse features imply how busy communication resources are. For example, instances demonstrating a large max-lapse values from the average-lapse values means that certain packets are sent from a client to a server with a larger delay than average. Lapse features do not require deep packet inspections, i.e. parsing the content of application data, thereby allowing analysis on encrypted traffic.

The traffic generated in the previous phase yielded 9 packet types, i.e.: application data, client hello, client key exchange, encrypted alert, SYN flag, ACK flag, RST flag, RST-ACK flag, and FIN-ACK. Each packet type was characterized by its 5 statistical properties (e.g. count, size, min lapse, average lapse, max lapse), yielding $9 \times 5 = 45$ features. However, there were no lapse values for SYN flag packets, because these packets identify connection initiation; the value of all 3 lapse features of SYN packets is always 0. The total number of features used in this study is therefore $45 - 3 = 42$ features.

A snapshot of traffic within an observed time window was characterised by the above features. In this study, the time window is one-second. Each one-second traffic represented an instance, and several instances comprised a dataset. Hence, tabular datasets were created with the rows as the instances of the traffic, and the columns as the features of each instance. One additional column, usually placed as the last one, contained nominal data that labelled the class of each instance, i.e. legitimate or attack. These datasets serve as inputs to the machine learning techniques that show how attack and legitimate traffic were classified.

4.2. Feature Ranking and Classification

As part of this phase, features were ranked and machine learning classification was analysed. Feature ranking reduces the number of inputs for machine learning processing and analysis, through finding and ranking the most relevant features. Ranked features aid the analysis of this study to find a set of features that can describe the characteristics of the traffic models. Two feature ranking

techniques were employed, i.e. Information Gain and Gain Ratio. *Information Gain* is a measure of purity when a feature is taken into account. Its value can be used to measure the degree of information if a new instance were classified as a certain class. The relevance value is given by equation (1),

$$Gain(feature) = Info(training) - Info(feature) \tag{1}$$

where $Info(training)$ is the amount of information when the whole set of training example is included, and $Info(feature)$ is the amount of information when a specific feature is selected. The amount of information is obtained from an entropy function that measures the degree of coherence with respect to a each class k, which is given in equation (2),

$$Info(x) = -\sum_{k=1}^{n} p_k log(p_k) \tag{2}$$

where p_k is the probability of an occurrence that an instance was classified as k when feature x is selected. In a two-class scenario as used in this study, the information of the training example set can be simplified as shown in equation (3).

$$Info(training) = -p_{legitimate} log(p_{legitimate}) - p_{attack} log(p_{attack}) \tag{3}$$

A shortcoming of Information Gain is that features with a large range of possible values return a near-zero entropy value. Consequently, the Gain value of the feature becomes greater than any other features causing it to be ranked higher without truly representing its relevance to the class of data sample.

Gain Ratio is a feature ranking measure that compensates the above drawback. It normalizes the Gain value of the training dataset with the entropy value of the feature's subsets, named the $IntrinsicValue$. Gain Ratio formula is given in equation (4).

$$GainRatio(feature) = \frac{Gain(feature)}{IntrinsicValue} \tag{4}$$

The Intrinsic Value represents the information value of the feature. It disregards any information about the class of the data sample. This is given in equation (5).

$$IntrinsicValue = -\sum_{v \in values(ftr)} \frac{|x \in S, value(x, ftr)|}{|S|} . log_2 \frac{|x \in S, value(x, ftr)|}{|S|} \quad (5)$$

where S is the number of instances in the training dataset, x is a sample of the training dataset, ftr is the selected feature to measure, $values(ftr)$ is a set of all possible values of the selected feature, and $value(x, ftr)$ is the value of the selected feature in sample x.

The drawback of Gain Ratio is that it can rank a less relevant feature high, due to the feature's low intrinsic value. Therefore, this study applied both Information Gain and Gain Ratio to take the advantages of each measure and to ascertain comparison of the relevance of features to the classification process.

Different sets of ranked features were investigated to observe performance of classifying attack and legitimate traffic. Traffic classification was studied through employing four machine learning techniques, i.e. Naïve Bayes, Decision Tree, JRip, and Support Vector Machines. This study used Weka [34] to rank features and run a range of machine learning algorithms.

To analyse the performance of these machine learning techniques, the machine learning False Alarms were measured. This is given in equation 6, which is the percentage of instances incorrectly classified out of the total number of the whole instances S. The False Positive FP is the percentage of legitimate traffic that the machine learning algorithm incorrectly identifies as attack traffic. The False Negative FN is the percentage of attacks that the machine learning algorithm incorrectly identifies as legitimate traffic.

$$\text{False Alarms} = \frac{FP + FN}{S} \times 100\% \quad (6)$$

5. Results and Analysis

5.1. Experimental Setup

All devices employed in this study were virtual devices, run on a desktop computer that serves the host machine. The host runs Windows 10 on Intel-i7 quad core with 64 GB RAM; and the virtual machine was VMware Player 12. The host machine could run the whole virtual machines required in this study without showing any signs of RAM exhaustion or CPU utilisation. The network connecting the clients and the servers was comprised of VMware machines. There was no bandwidth limit set on the virtual network.

The *client* was used to generate traffic. Various traffic patterns, attacks as well as normal traffic, were generated and studied to model several client-server scenarios. Each client ran a Debian Linux Operating System, Ubuntu 15.04, on the VMware virtual machine.

The *server* was an HTTP/2 web server. This study used a publicly available HTTP/2 server, named `libevent-server` written by the nghttp2 author [31]. The investigation monitored the server for effects of resource consumption. In order to detect symptoms of resource consumption, the following measurements were to be monitored [35]: CPU usage, memory consumption, network throughput, and packet loss. When a computing resource is under load, CPU usage, memory consumption and packet loss indicators can show increasing activities, while network throughput can decrease.

5.2. Statistical Values of the Mimicked Features

The Stealthy Attack-1 model attempted to mimic the count_syn feature values, while the Stealthy Attack-2 model aimed to mimic the size_rstAck feature values. Figure 6 shows the visualisation of these stealthy attack feature values when compared to those of legitimate traffic. In both figures a and b, the black graph represents the feature values of the legitimate traffic, while the grey graph shows those of the attack traffic. It can be seen that the Stealthy Attack-1 count_syn feature values amalgamate with those of the legitimate traffic (Figure

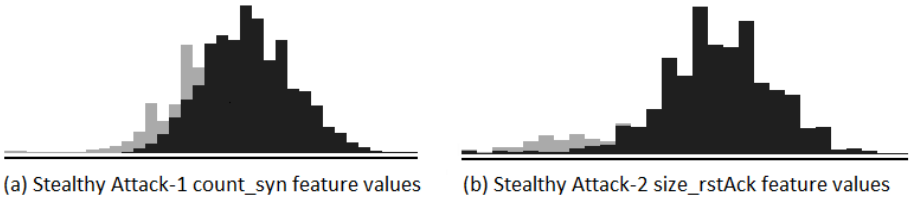


Figure 6: Visualisation of the mimicked feature values

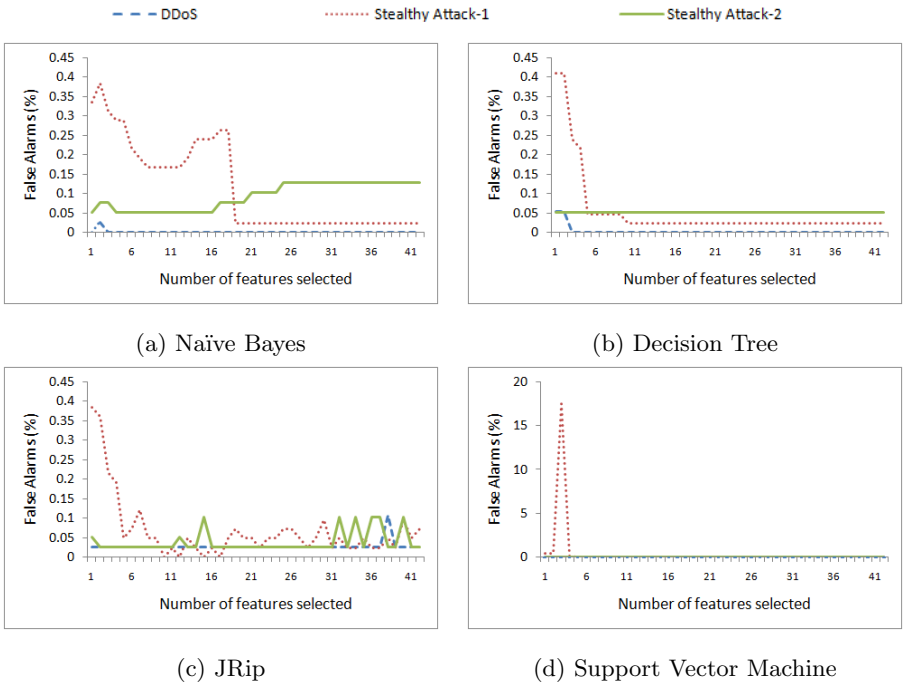
6.a). Similarly, the Stealthy Attack-2 size_rstAck feature values mimicked those of the legitimate traffic (Figure 6.b).

The figure illustrates that it is not trivial to find a threshold value that separates the attack and legitimate traffic. This means intrusion-detection systems that depends on a predefined threshold value to separate attack and legitimate traffic would produce some False Alarms. This finding contrasts with the HTTP/2 DDoS traffic detection, where a threshold value can be set to separate DDoS and legitimate traffic, as shown earlier in Figure 4. Therefore, the stealthy attack traffic is stealthier than the DDoS traffic.

5.3. Analysis using Machine Learning Techniques

While the previous discussion analyses attack detection using specific features from the stealthy attack models, this subsection uses the proposed 42 features to analyse the number of False Alarms. Four machine learning techniques, i.e. Naïve Bayes Decision Tree, JRip and Support Vector Machines, were employed to yield the percentage of False Alarms when attack and legitimate traffic were analysed. The machine learning classifications were executed on an Intel Core-i3 machine with a 2 GB RAM. Three of the machine learning techniques, i.e. Naïve Bayes, Decision Tree and JRip, took less than 1 second to yield classification results. In contrast, Support Vector Machines took more than 2 minutes to yield classification results.

The study employed Information Gain technique to rank features. The technique produced a range of feature set X, where the most relevant $[1, x]$ features were employed for machine learning analysis. Figure 7 shows the performance of



(a) Naïve Bayes (b) Decision Tree

(c) JRip (d) Support Vector Machine

Figure 7: Visual inspection to find a range of X-values that yield low False Alarms values.

machine learning techniques in terms of False Alarms (Y-axis), for each feature set $[1, x]$, where $x = 1 \ldots 42$.

Visual inspections were applied to find a range of numbers of selected features, to yield a low percentage of False Alarms. The reasoning behind this is that intrusion-detection systems would choose rules that can yield the lowest False Alarms value. Figure 7 was used to visually choose a number of selected features. The figure combined the False Alarms (Y-axis) of the 3 attack traffic models.

From visually inspecting Figure 7.a, a hypothetical intrusion-detection-system that employed Naïve Bayes would choose 19 selected features or higher to obtain low Y values. In this regard, Stealthy Attack-2 was found to be the stealthiest due to its high Y values, followed by Stealthy Attack-1 and DDoS.

A visual inspection on Decision Tree (Figure 7.b) also shows that Stealthy

Attack-2 yielded the highest Y values, hence the stealthiest, when a hypothetical intrusion-detection-system employed 5 Information Gain-ranked features or higher. The figure also shows that Stealthy Attack-1 was less stealthy, and DDoS was the least stealthy.

JRip (Figure 7.c) graphs show that between 10 and 17 features can be selected to yield low Y values. When these features are selected, Stealthy Attack-2 was the stealthiest (as it showed the highest Y value), followed by DDoS and Stealthy Attack-1.

Support Vector Machine (Figure 7.d) show otherwise. When 5 or more features were employed, DDoS was the stealthiest, as it yielded 0.024% False Alarms. A hypothetical intrusion-detection-system can distinguish the other two attack traffic models, Stealthy Attack-1 and Stealthy Attack-2, from flash-crowd when 5 or more features were employed.

From the visual inspection in this subsection, Naïve Bayes, Decision Tree, and JRip show that Stealthy Attack-2 bears the stealthiest traffic among the 3 traffic models analysed. Support Vector Machine yield otherwise, with DDoS being the stealthiest.

5.4. Boundary Values

This subsection extends the results obtained from the previous analysis to examine the boundary values. The boundary performance values were indicated by the best figure a classifier measured regardless of the number of features selected, i.e. the farthest Y value regardless of X. Hence, these were the lowest values of False Alarms from each traffic types. This represents the best result machine learning techniques can be employed to distinguish attack from legitimate traffic. A higher boundary value signifies stealthier traffic. The intuition behind this is that attackers would choose attack traffic models that yield higher False Alarms, given the best detection method was known.

Furthermore, this subsection extends the previous analysis to include Gain Ratio feature ranking technique. The result is shown in Figure 8. The figure shows that Stealthy Attack-2 traffic (SA-2) was the stealthiest when measured

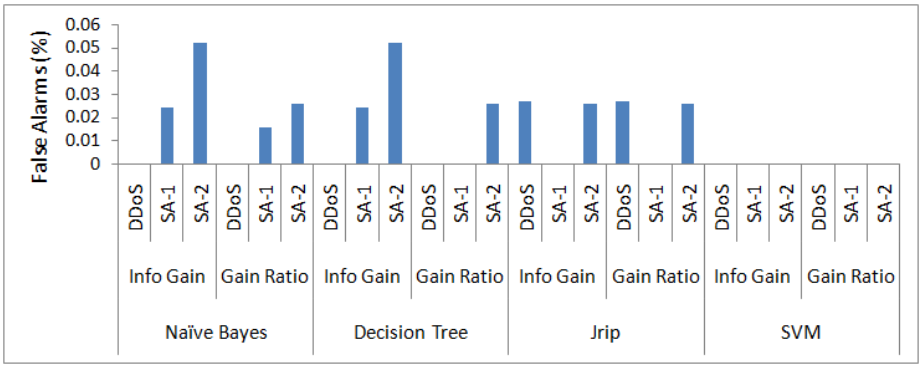


Figure 8: False Alarms for all 3 attack types

with Naïve Bayes and Decision Tree. However, this result was not uniform across other classifiers tested. JRip showed that DDoS traffic was stealthier than Stealthy Attack-2, as DDoS showed a higher bar than Stealthy Attack-2. Similarly, JRip showed that Stealthy Attack-2 was stealthier than Stealthy Attack-1 (SA-1), as the former showed a higher bar than the latter. Support Vector Machines were able to yield 0% False Alarms when classifying the 2 proposed attack traffic models and the DDoS traffic.

It has been shown that the two attack traffic models presented in this study can be identified using Support Vector Machines. On the other hand, they produced higher percentages of False Alarms than the HTTP/2 DDoS attack traffic, when analysed using Naïve Bayes, Decision Tree, and JRip. Thus, the stealthy attack models yield stealthier traffic than the DDoS traffic.

5.5. Feature Ranking Comparison with HTTP/1.1 Features

Current research on DDoS attack classification explores methods to distinguish attack traffic from legitimate or flash-crowd traffic operating in an HTTP/1.1 environment. In contrast, this study operated in HTTP/2 environment to classify legitimate from attack traffic. This subsection also discusses how the HTTP/1.1 features can be ranked differently as distinguishing factors when applied to HTTP/2 traffic.

To make a comparison, the study investigated features that were used for

both HTTP/1.1 and HTTP/2 traffic analysis. Listing these common features was a non-trivial task, because the traffic patterns varied significantly. HTTP/1.1 traffic analysis was reliant on unencrypted packets, while HTTP/2 analysis carried out in this study was applied to encrypted traffic. HTTP/1.1 traffic analysis required deep packet inspection, where unencrypted HTTP packets such as the content of HTTP Requests [36, 37, 38, 39], its distribution [40], or flow [41] could be analysed. Similarly, the number of resources (i.e. files or pages) that clients requested from a server was one of the features to model flash-crowd traffic [42]. Again, these solutions relied on unencrypted HTTP data.

As a result, many of the previous methods that inspected HTTP/1.1 traffic were no longer applicable when implemented for encrypted traffic analysis of HTTP/2. This is because encrypted traffic conceals the content of the application-layer data. Hence, browsing behaviour that depended on HTTP data and packet flow measurements, and required application-data inspection could not be analysed through these techniques. Consequently, deep packet inspection methods and features that relied on certain HTTP/1.1 packet flow observations as described above could not be applied to analyse encrypted HTTP/2 traffic.

However, certain aspects were adoptable from HTTP/1.1 traffic analysis and applied to HTTP/2 traffic. Both HTTP protocols required IP headers to deliver client to server HTTP messages. The IP header was not commonly encrypted in HTTP/2 unless for tunnelling purposes (where client to server IP headers were encrypted to obscure client-server IP addresses). Hence, IP header information was valuable analysis. Traditionally, studies observed the entropy of the IP address and the port number as the distinguishing features to detect DDoS attacks against HTTP/1.1 services [43, 44]. The disadvantage of this solution was that IP address spoofing, or forging the source IP address, has increasingly been a common practice adopted by adversaries. Attackers could mimic the statistical properties of legitimate traffic and bypass detection methods. Therefore, these features alone are insufficient for categorizing DDoS attack traffic.

A more recent approach [45] applied statistical components of the network headers, i.e. the number of connections and the number of packets, to define a determining factor using joint-entropy. The method showed high coherence between any two factors in a flash-crowd event, while attacks yielded a deviation from some values that indicated coherency. The study clarified that it only inspected the IP header field which did not require deep packet inspections. However, as previously explained, IP header values can be spoofed and the distribution of the values could be made to mimic flash-crowd properties, making it harder to distinguish.

The features used in the above HTTP/1.1 DDoS detection methods that inspected IP headers could be applied to compare the stealthiness of the HTTP/2 attack traffic proposed in this study. These HTTP/2 features were the number of Application Data packets in 1-second traffic instances, or the *count_app* features, and the number of packets carrying SYN flags, or the *count_syn* features, representing the two HTTP/1.1 features, i.e. the number of packets and the number of connections, respectively.

Other than these sets of features used for the study, the HTTP/2 traffic analysis in this study applied machine learning techniques, while the previous study [45] used joint-entropy. In this case, the advantage of using machine learning is that a range of features could be compared and ranked according to their relevance to detecting and differentiating attack traffic.

The results are presented in Tables 4. The two HTTP/1.1 traffic features are considered at the top of the rank because they were the most distinguishing factors for distinguishing HTTP/1.1 DDoS from flash-crowd traffic. When applied to analyse HTTP/2 traffic, it can be seen that the count_syn feature deviated significantly from the top ranked feature. This was true for both Information Gain and Gain Ratio-based feature ranking.

Furthermore, Table 4 shows that the count_app feature, remained a highly relevant feature as it was still ranked near the top for all three HTTP/2 traffic types, i.e. the DDoS, Stealthy Attack-1 (SA-1) and Stealthy Attack-2 (SA-2). This confirmed that the Denial-of-Service attack demonstrated in this study was

Table 4: Feature ranks for all 3 attack types

Feature	HTTP/1.1	HTTP/2					
		Information Gain			Gain Ratio		
		DDoS	SA-1	SA-2	DDoS	SA-1	SA-2
count_app	top (1 and 2)	1	3	3	1	5	4
count_syn		4	8	13	6	7	17

categorized accurately as a flooding-based attack, and the HTTP/2 server was incapacitated due to a high traffic volume from the attack.

This led to further discussions on the external validity of the outcomes achieved. Hence, a comparison with other studies was made to justify how the standard traffic model used in this study, i.e. the synthetically generated legitimate traffic, can be differentiated from real flash-crowd traffic. For example, the standard traffic used might be too dense because the legitimate traffic was generated using 5,200 clients (Subsection 3.1.3) while another study used only 80 clients to generate similar traffic volume [41]. When the HTTP/2 attack traffic was compared to a network with such low traffic density, the attack traffic would yield a much higher number of packets and therefore could be easily distinguishable from legitimate. However, the study that used 80 clients also used simulated data; hence, the low number of clients did not mimic real traffic accurately.

Currently there is no real HTTP/2 dataset available. The closest publicly available HTTP/1.1 dataset that describes flash-crowd traffic is the World Cup 98 [46]. The data from this dataset was based on more than 3,000 client requests generated per second. Assuming user-browsing time was 15 seconds as in what can be observed from the DOBBS log (Section 3.1.2), the number of clients in this dataset was equal to 45,000. This number was almost 9 times higher than the 5,200 clients used to generate legitimate traffic in this study. If real HTTP/2 traffic characteristics were similar to that, the stealthy traffic proposed in this study would cause the count_app feature to become less relevant. Therefore,

Table 5: False Alarms (%) by machine learning techniques applied with only HTTP/1.1 features.

	Stealthy Attack 1	Stealthy Attack 2
Naïve Bayes	0.2651	0.5189
Decision Tree	0.1687	0.2335
JRip	0.1205	0.2335
Support Vector Machine	0	0.3373

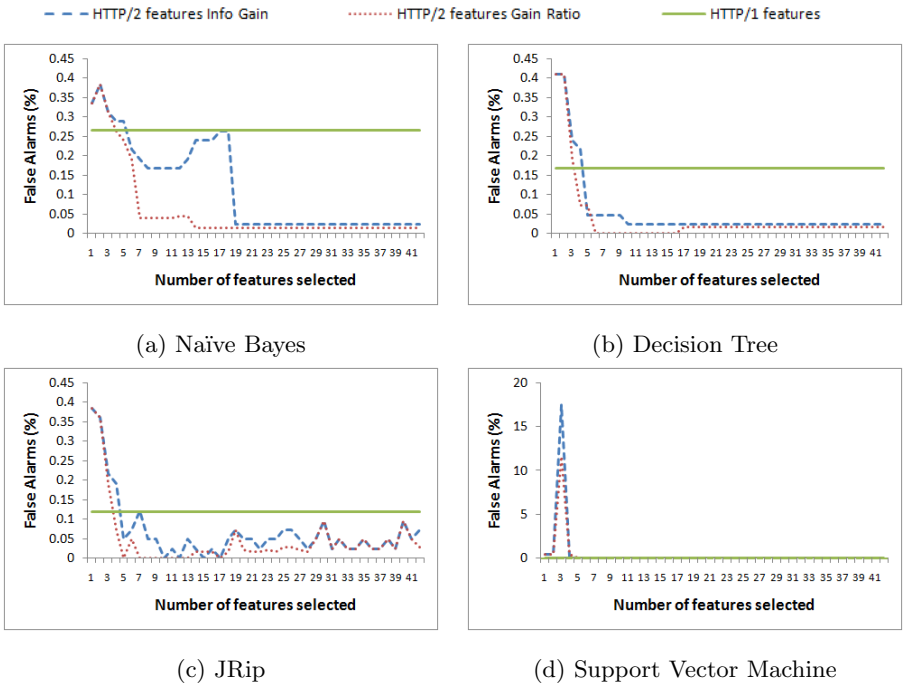
when a machine learning technique is applied to classify legitimate traffic, the stealthy traffic could yield more stealthy properties such as a higher number of False Alarms.

5.6. Performance Comparison

While the previous subsection identified features used in HTTP/1.1 as determining factors for differentiating legitimate traffic from DDoS attack traffic, this subsection presents how machine learning techniques perform when the traffic was analysed using only two HTTP/1.1 features, i.e. the *count_app* and *count_syn* features. Furthermore, it compared the machine learning technique performance analysis when using the two HTTP/1.1 features to the 42 HTTP/2 features proposed in this study. The study examined the False Alarms as an evaluation measure.

The machine learning-based analysis results, when using the HTTP/1.1 features as determining factors, is shown in Table 5. It can be seen that Stealthy Attack-2 yielded higher percentages of False Alarms than Stealthy Attack-1. This was another evidence that Stealthy Attack-2 proved to be stealthier than Stealthy Attack-1.

The methodologies for comparison of the results with the presented attack models (Section 3.2) are shown in Figure 9 and 10. The former is a comparison to the Stealthy Attack-1 performance, while the latter is a comparison to the Stealthy Attack-2 performance. In the two figures, classification performance using HTTP/1.1-features from Table 5 is shown as a straight line in all graphs



(a) Naïve Bayes

(b) Decision Tree

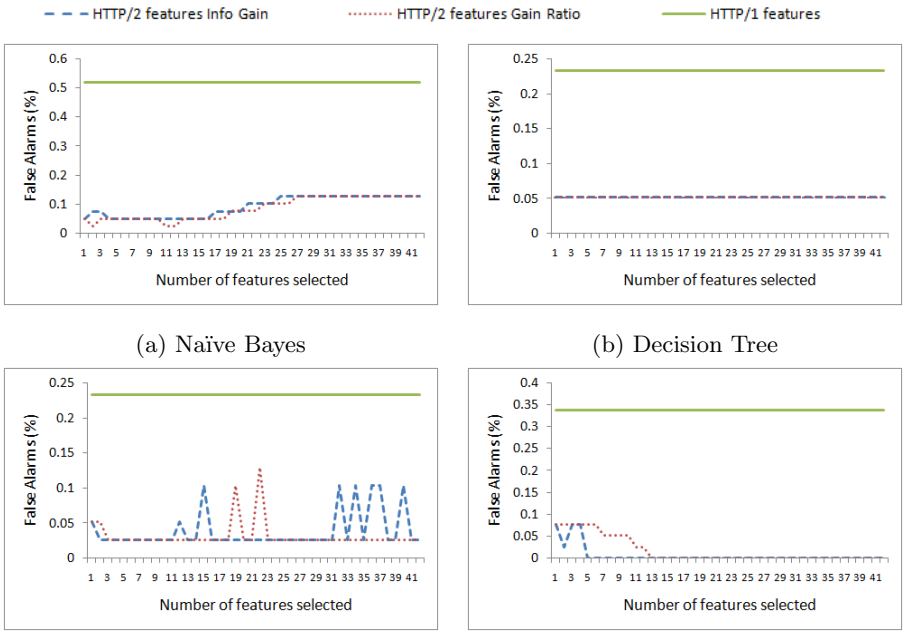
(c) JRip

(d) Support Vector Machine

Figure 9: A comparison of Stealthy Attack-1 performance when using both HTTP/1.1 features and HTTP/2 features.

to serve as a baseline value. The baseline value was to visualize gaps to classification performance results using HTTP/2-features, illustrated by the dotted curves in the figures. It can be seen from the figures that the straight lines are higher, away from the X-axis, than the dotted curves. This means that classifying the Stealthy Attack-1 and Stealthy Attack-2 from flash crowd yielded more False Alarms when employing HTTP/1.1 features than when employing the HTTP/2 features proposed in this study. The HTTP/2 features yielded better results than the HTTP/1.1 features, when they were employed by machine learning techniques to distinguish HTTP/2 attack traffic from flash crowd traffic.

Another observation drawn from Figures 9 and 10 is that Stealthy Attack-2 is stealthier than Stealthy Attack-1. When the straight lines were set as a baseline, larger gaps between the line and the dotted curves are clearly seen on the



(a) Naïve Bayes

(b) Decision Tree

(c) JRip

(d) Support Vector Machine

Figure 10: A comparison of Stealthy Attack-2 performance when using both HTTP/1.1 features and HTTP/2 features.

Stealthy Attack-2 figures (Figure 10). Therefore, when a hypothetical intrusion-detection system, equipped with a classifier employing HTTP/1.1 features was used to measure the False Alarms of the two traffic models proposed in this study, Stealthy Attack-2 yielded stealthier traffic than Stealthy Attack-1.

Two conclusions can be drawn from the analysis and discussions in this subsection. First, the 42 HTTP/2 features proposed in this study were able to yield better results than the HTTP/1.1 features proposed in the literature, in distinguishing attack from legitimate traffic. This observation was true in all four cases, i.e. when the analysis used Naïve Bayes, Decision Tree, JRip, and Support Vector Machines. This demonstrates that the HTTP/2 features proposed in this study yielded better results than the HTTP/1.1 features employed in the literature. Second, the stealthy attack model proposed in this study can be implemented to yield attack traffic that demonstrate more stealthy properties

to mimic legitimate traffic features. It was demonstrated that Stealthy Attack-2 traffic yield higher False Alarms when analysed using HTTP/1.1 features.

6. Conclusion & Future Work

This paper presents two HTTP/2 stealthy attack models that operate by intelligently distributing network traffic load associated with a Denial of Service (DoS) attack over an extended period of time, to remain covert. The stealthy traffic affected the performance of machine learning classifiers to yield higher percentages of False Alarms. To show how the stealthy attack traffic were classified from legitimate traffic, this study also presents an HTTP/2 legitimate traffic model to generate flash-crowd traffic. Furthermore, this study presented distinguishing factors for traffic identification through a set of network traffic features to characterise the generated stealthy attack traffic and flash-crowd traffic.

The study showed how features of HTTP/1.1 traffic were not highly relevant when used to analyse HTTP/2 traffic. Instead, the analysis of the two HTTP/2 stealthy attack models, Stealthy Attack-1 and Stealthy Attack-2, performed better when the proposed HTTP/2 features were used as the distinguishing factors, as machine learning classifiers yielded less percentage of False Alarms when applying HTTP/2 features than HTTP/1.1 features. It was shown that the stealthy attack models caused machine learning analysis to yield higher percentage of False Alarms than the DDoS model. This showed that the proposed stealthy attack models can bypass intrusion-detection systems that employed machine learning techniques.

Future work in HTTP/2 DoS attack design and analysis can benefit from actual HTTP/2 flash-crowd traffic. This study has contributed in creating novel, HTTP/2 legitimate and attack datasets. As Internet-connected devices and its heterogeneity are projected to increase significantly in the future, research focusing on creating, collecting and synthesizing current Internet traffic datasets will continue to extend knowledge published through this research work. On the

other hand, HTTP/2 introduces new techniques capable of generating various traffic patterns, which have been shown in this study to produce DoS attack traffic of varying characteristics. These can lead to further investigation on DoS attack design, analysis, and detection for HTTP/2 services.

References

[1] I. Grigorik, Making the web faster with HTTP 2.0, Communications of the ACM 56 (12) (2013) 42–49.

[2] M. Belshe, R. Peon, M. Thomson, Hypertext Transfer Protocol version 2 (HTTP/2), Report RFC 7540, Internet Engineering Task Force (IETF) (May 2015).

[3] S. Heron, Denial of Service: Motivations and trends, Network Security 2010 (5) (2010) 10–12.

[4] S. Mansfield-Devine, DDoS: Threats and mitigation, Network Security 2011 (12) (2011) 5–12.

[5] CERT, Denial of service attacks, accessed: 2016-09-06 (1997).
URL `https://www.cert.org/information-for/denial_of_service.cfm?`

[6] S. T. Zargar, J. Joshi, D. Tipper, A survey of defense mechanisms against distributed denial of service (DDoS) flooding attacks, Communications Surveys & Tutorials, IEEE 15 (4) (2013) 2046–2069.

[7] R. K. Chang, Defending against flooding-based Distributed Denial-of-Service attacks: a tutorial, Communications Magazine, IEEE 40 (10) (2002) 42–51.

[8] C. Labovitz, The Internet goes to war (December 14 2010).
URL `http://www.arbornetworks.com/asert/2010/12/the-internet-goes-to-war/`

[9] P. Paganini, Dangerous DDoS (Distributed Denial of Service) on the rise (May 28 2013).
URL http://resources.infosecinstitute.com/dangerous-ddos-distributed-denial-of-service-on-the-rise/

[10] J. Keane, DDoS attack hit record numbers in Q2 2015, Digital Trends (Aug. 2015).
URL http://www.digitaltrends.com/computing/ddos-attacks-hit-record-numbers-in-q2-2015/

[11] S. Khandelwal, 602 Gbps! This may have been the largest DDoS attack in history, The Hacker News (Jan. 2016).
URL http://thehackernews.com/2016/01/biggest-ddos-attack.html

[12] E. Adi, Z. Baig, C. P. Lam, P. Hingston, Low-rate denial-of-service attacks against HTTP/2 services, in: IT Convergence and Security (ICITCS), 2015 5th International Conference on, IEEE, 2015, pp. 1–5.

[13] E. Adi, Z. A. Baig, P. Hingston, C. P. Lam, Distributed denial-of-service attacks against HTTP/2 services, Cluster Computing 19 (1) (2016) 79–86. doi:10.1007/s10586-015-0528-7.

[14] X. Wu, V. Kumar, J. R. Quinlan, J. Ghosh, Q. Yang, H. Motoda, G. J. McLachlan, A. Ng, B. Liu, S. Y. Philip, Top 10 algorithms in data mining, Knowledge and Information Systems 14 (1) (2008) 1–37.

[15] A. W. Moore, D. Zuev, Internet traffic classification using bayesian analysis techniques, in: ACM SIGMETRICS Performance Evaluation Review, Vol. 33, ACM, 2005, pp. 50–60.

[16] S. Mukherjee, N. Sharma, Intrusion detection using naive bayes classifier with feature reduction, Procedia Technology 4 (2012) 119–128.

[17] J. Zhang, C. Chen, Y. Xiang, W. Zhou, Y. Xiang, Internet traffic classification by aggregating correlated naive bayes predictions, Information Forensics and Security, IEEE Transactions on 8 (1) (2013) 5–15.

[18] F. Haddadi, J. Morgan, A. N. Zincir-Heywood, Botnet behaviour analysis using IP flows: With HTTP filters using classifiers, in: Advanced Information Networking and Applications Workshops (WAINA), 28th International Conference on, IEEE, 2014, pp. 7–12.

[19] K. Swamy, K. V. Lakshmi, Network intrusion detection using improved decision tree algorithm, IJCSIS) International Journal of Computer Science and Information Security 10 (8).

[20] W. W. Cohen, Fast effective rule induction, in: Proceedings of the Twelfth International Conference on Machine Learning, 1995, pp. 115–123.

[21] W. N. H. W. Mohamed, M. N. M. Salleh, A. H. Omar, A comparative study of reduced error pruning method in decision tree algorithms, in: Control System, Computing and Engineering (ICCSCE), 2012 IEEE International Conference on, IEEE, 2012, pp. 392–397.

[22] P. Gaonjur, N. Tarapore, S. Pukale, M. Dhore, Using neuro-fuzzy techniques to reduce false alerts in IDS, in: 2008 16th IEEE International Conference on Networks, IEEE, 2008, pp. 1–6.

[23] J. Yang, A. Tiyyagura, F. Chen, V. Honavar, Feature subset selection for rule induction using RIPPER, in: Proceedings of the Genetic and Evolutionary Computation Conference, Vol. 2, 1999, p. 1800.

[24] M. Panda, A. Abraham, M. R. Patra, Hybrid intelligent systems for detecting network intrusions, Security and Communication Networks 8 (16) (2015) 2741–2749.

[25] V. N. Vapnik, V. Vapnik, Statistical Learning Theory, Vol. 2, Wiley New York, 1998.

[26] A. K. Sharma, P. S. Parihar, An effective DoS prevention system to analysis and prediction of network traffic using support vector machine learning, International Journal of Application or Innovation in Engineering & Management 2 (7) (2013) 249–256.

[27] Z. Li, R. Yuan, X. Guan, Accurate classification of the Internet traffic based on the SVM method, in: Communications, 2007. ICC'07. IEEE International Conference on, IEEE, 2007, pp. 1373–1378.

[28] C. von der Weth, M. Hauswirth, DOBBS: Towards a comprehensive dataset to study the browsing behavior of online users, in: Web Intelligence (WI) and Intelligent Agent Technologies (IAT), 2013 IEEE/WIC/ACM International Joint Conferences on, Vol. 1, IEEE, 2013, pp. 51–56.

[29] G. Combs, Wireshark (version 2.0.1) (1998–2015).
URL `https://www.wireshark.org/`

[30] S. Yu, S. Guo, I. Stojmenovic, Can we beat legitimate cyber behavior mimicking attacks from botnets?, in: INFOCOM, 2012 Proceedings IEEE, IEEE, 2012, pp. 2851–2855.

[31] T. Tsujikawa, Nghttp2: HTTP/2 C library (2015).
URL `https://nghttp2.org/`

[32] D. Stenberg, cURL (1996–2016).
URL `https://curl.haxx.se/download.html`

[33] S. Bradner, Key words for use in RFCs to indicate requirement levels, Report RFC 2119, Network Working Group (1997).

[34] University of Waikato, Weka (version 3.8) (1993–2016).
URL `http://www.cs.waikato.ac.nz/ml/weka/downloading.html`

[35] K. Salah, K. Sattar, M. Sqalli, E. Al-Shaer, A potential low-rate DoS attack against network firewalls, Security and Communication Networks 4 (2) (2011) 136–146.

[36] J. Jung, B. Krishnamurthy, M. Rabinovich, Flash crowds and Denial of Service attacks: Characterization and implications for CDNs and web sites, in: Proceedings of the 11th International Conference on World Wide Web, ACM, 2002, pp. 293–304.

[37] G. Oikonomou, J. Mirkovic, Modeling human behavior for defense against flash-crowd attacks, in: Communications, 2009. ICC'09. IEEE International Conference on, IEEE, 2009, pp. 1–6.

[38] M. A. Saleh, A. Abdul Manaf, A novel protective framework for defeating HTTP-based denial of service and distributed denial of service attacks, The Scientific World Journal 2015.

[39] W. Zhou, W. Jia, S. Wen, Y. Xiang, W. Zhou, Detection and defense of application-layer DDoS attacks in backbone web traffic, Future Generation Computer Systems 38 (2014) 36–46.

[40] S. Bhatia, G. Mohay, A. Tickle, E. Ahmed, Parametric differences between a real-world distributed denial-of-service attack and a flash event, in: Availability, Reliability and Security (ARES), 2011 Sixth International Conference on, IEEE, 2011, pp. 210–217.

[41] M. Sachdeva, K. Kumar, A traffic cluster entropy based approach to distinguish DDoS attacks from flash event using DETER testbed, ISRN Communications and Networking 2014.

[42] S. Bhatia, G. Mohay, D. Schmidt, A. Tickle, Modelling web-server flash events, in: Network Computing and Applications (NCA), 2012 11th IEEE International Symposium on, IEEE, 2012, pp. 79–86.

[43] K. Kumar, R. Joshi, K. Singh, A distributed approach using entropy to detect DDoS attacks in ISP domain, in: Signal Processing, Communications and Networking, 2007. ICSCN'07. International Conference on, IEEE, 2007, pp. 331–337.

[44] A. Lakhina, M. Crovella, C. Diot, Mining anomalies using traffic feature distributions, in: ACM SIGCOMM Computer Communication Review, Vol. 35, ACM, 2005, pp. 217–228.

[45] H. Rahmani, N. Sahli, F. Kamoun, Distributed Denial-of-Service attack detection scheme-based joint-entropy, Security and Communication Networks 5 (9) (2012) 1049–1061.

[46] WorldCup98 dataset (1998).
URL http://ita.ee.lbl.gov/html/contrib/WorldCup.html