ADDIS ABABA UNIVERSITY

ADDIS ABABA INSTITUTE OF TECHONOLOGY

SCHOOL OF ELECTRICAL & COMPUTER ENGINEERING

Application Layer DDoS Attack Detection In The Presence Of Flash Crowds

By: Biruk Asmare

A Thesis Submitted To School Of Graduate Studies

In Partial Fulfillment For The

Degree Of Master Of Science In Computer Engineering

Thesis Advisor Dr. Surafel Lemma

Chairman of Department

Dr. Yalemzewud Negash

Thesis Evaluator

Thesis Evaluator

Thesis Evaluator

Application Layer DDoS Attack Detection In The Presence Of Flash Crowds

By

Biruk Asmare

MSC Thesis

Submitted in partial fulfillment of the requirements

for master of science in computer engineering

Addis Ababa Institute Of Technology

School Of Electrical & Computer Engineering

September, 2017

Declaration

This thesis is a presentation of my original research work. Wherever contributions of others are involved, every effort is made to indicate this clearly, with proper citation of sources. I, the undersigned, declare that this thesis has not been presented for a degree in any other university.

Biruk Asmare

This thesis has been submitted for examination with my approval as a university advisor.

Dr. Surafel Lemma

Acknowledgement

First, I want to thank Jesus Christ and St. Mary who guide me on every aspect of my life. I give my deepest gratitude to my advisor Dr. Surafel Lemma for his continuous support and supervision throughout the whole duration of this work. I also thank all instructors who gave me their invaluable comments during my seminars. I am also grateful to my family, specially My mother Abynesh Tadele, for their continuous support and encouragement. I also like to thank Ageritu Kassa for her support on statistical analysis of data and results. Last but not least, I would like to thank Abere Abebe, Amelework Asmare, Meskerem Asmare, Bazen Gashaw and Dagimawi Demessie who supported and encouraged me.

Abstract

Application layer DDoS attacks are growing at alarming rate in terms of attack intensity and number of attack. Attackers target websites of government agencies as well as private business for different motives. One particular research problem is distinguishing Application layer DDoS attacks from flash crowds. Both flash crowds and application layer DDoS attack cause denial of service. Flash crowds come from sudden surge in traffic of legitimate requests. Whereas, application layer DDoS attacks are intentionally generated by attackers to cause denial of service. Distinguishing between Application layer DDoS attacks and flash crowd is important because the action taken to address both problems is different. Flash crowds are legitimate requests which should be serviced. Whereas, Application layer DDoS attacks are malicious requests that should not be serviced. Furthermore, the source of application layer DDoS attacks should be blocked from making further requests. In this research, supervised machine learning based application layer DDoS detection approach was proposed to distinguish between application layer DDoS attack and flash crowd. Features that help distinguish application layer DDoS attacks from legitimate flash crowds were identified. Six supervised classifiers were evaluated using World cup 98 flash crowd dataset and experimentally generated application layer DDoS attack dataset. We have selected decision tree as supervised classifier in our detection system based on evaluation result. Decision tree had F1 score of 99.45% and False positive rate of 0.47%.

keywords:

APP-DDoS attack, application layer, decision tree classifier, flash crowd, layer7 attacks, supervised machine learning

Contents

List of Figures

List of Tables

Acronyms

ANOVA	Analysis Of Variance
APP-DDoS	Application layer Distributed Denial of Service
CART	Classification And Regression Tree
DoS	Denial of Service
DDoS	Distributed Denial of Service
DR	Download Rate
DT	Decision Tree
FN	False Negative
FP	False Positive
FPR	False Positive Rate
FTP	File Transfer Protocol
GNB	Gaussian Naive Bays
HTTP	Hypertext Transfer Protocol
ID3	Iterative Dichotomiser 3
IP	Internet Protocol
PP	Page Popularity
RIA	Request Inter-arrival
RR	Request Rate
RSR	Ratio of Successful Requests
SMTP	Simple Mail Transfer Protocol
SVM	Support Vector Machine
SVML	SVM with linear kernel
SVMP	SVM with polynomial kernel
SVMR	SVM with radial basis kernel
TCP	Transmission Control Protocol
TN	True Negative

TP	True Positive
URI	Uniform Resource Identifier
URL	Uniform Resource Locator

Chapter 1

Introduction

DoS attacks are attacks against availability of Internet service. DoS attack is characterized by an attempt to prevent legitimate users from using web services by degrading server resources. DoS attacks are becoming major Internet security threats (Akamai, 2016). The intentions of the attackers range from creating inconvenience to users to some serious financial damage to companies that are dependent on online business.

DoS attacks are also used as a facilitator for other attacks. This is performed by first attacking the legitimate web server and then creating a fake web server to deceive users to access their personal information.

Attacks that create DoS could be generated from multiple sources. Such DoS is referred to as **distributed denial of service(DDoS) attack**. DDoS attack utilizes multiple geographically distributed computer systems that are compromised or intentionally used to generate attack packets. The first documented DDoS attack was targeting major websites including Yahoo!, Amazon and eBay on February 7 (Norman, 2016). DDoS attacks are often used to promote some political agenda and as a means of cyber war. Most government organizations and large corporate websites are often targeted by groups that try to enforce their own agenda.

The openness of network communication protocols allowed easy exploit for attackers by identifying the vulnerability of different communication layer protocols. For example, during TCP three-way handshake, a client first sends TCP sync packet to server, the server allocates resources and sends acknowledge packet to the client. Then waits for acknowledgement from the client. SYN-flooding DDoS attack works by first sending the sync packet and not sending acknowledge packet after the server sends sync acknowledge packet. The server has to wait for some time before terminating the connection. So with

enough Sync packets, the server resources can be exhausted easily.

DDoS attacks are divided in to application layer and network layer attacks. Most common DDoS attacks are network layer attacks that try to exploit network resource and bandwidth of target host. Application layer DDoS attack uses application layer protocols such as HTTP, FTP and SMTP. The attack is conducted after creating a successful TCP connection which makes the attacks resistant to most network layer detection and mitigation systems. As a result, it is difficult to detect and mitigate Application layer DDoS attacks.

Application layer DDoS attacks are classified in to *request flooding attacks, asymmetric attacks and repeated one shot attacks* (Ranjan et al., 2006). In request flooding attack, attacker sends application layer requests such as HTTP GET request in higher rate than normal. In asymmetric attacks, attacker uses requests that require high workload on the server and by making multiple requests, the attacker easily crushes the server. In repeated one shot attack, the attacker sends requests that require high server workload in multiple secessions to avoid detection.

Although research on Application layer DDoS attack is young. A number of methods were proposed to help distinguish between Application layer DDoS attack and normal activity (Xie and Yu, 2006; Ranjan et al., 2006; Wang et al., 2010; Patil and Kulkarni, 2011; Xu et al., 2014; Yadav and Selvakumar, 2015). (Wang et al., 2010) proposed relative entropy based detection method. (Yadav and Selvakumar, 2015) proposed a detection approach that models legitimate users' behavior using logistic regression.

In this research, we deal with application layer DDoS attacks that occur together with flash crowds. *Flash crowd is a sudden or anticipated rise in the number of requests to a website by legitimate clients due to the addition of some news or when a new product is released.* Application layer DDoS attacks have similar characteristics with legitimate flash crowds. The proper reaction taken against flash crowd is to increase the server resources. Whereas, the action taken against Application layer DDoS attacks is to identify and

terminate attack requests. Distinguishing between flash crowds and App DDoS attacks is a very important network security problem.

We propose a detection system that identifies Application layer DDoS attacks from legitimate flash crowds. The core part of the detection system is a supervised learning classifier that classifies a client to either normal client or attack client. A client is a machine which is identified by an IP address and makes a request to a web server. The classifier is trained using examples of both flash crowds and DDoS attack. Information obtained from web server logs is used as an input to the classifier. We have identified five features, by studying related research, that will help identify DDoS attack from flash crowd. Each feature is computed by considering a certain time interval, called session time, for each unique client.

As a case study, we used World cup 98 dataset (Arlitt and Jin, 1998) to model flash crowds. It is a collection of requests made to *www.france98.com* during the duration of world cup 98 football game. World cup 98 dataset is used as flash crowd dataset in related researches (Xie and Yu, 2009; Yu et al., 2012; Bhandari et al., 2016). We prepared Application layer DDoS attack dataset by performing attack on locally hosted version of *www.france98.com* using BoNeSi (Markus-Go, 2016) DDoS attack tool.

In the case study, we tried to address three research questions related to application layer DDoS attack detection in the presence of flash crowds. The first question is investigation of the contribution of identified features for detection. The second research question focuses on evaluation of the proposed detection approach. The performance of the proposed detection approach is determined by the performance of the supervised classifier used for detection. We used F1 score, False Positive Rate (FPR) and classification time as evaluation criteria. Classification time is used to measure the computational efficiency of the classifiers. The third research question addresses the effect of session time on detection performance. Session time is important because it determines the response time of the detection system. Lower session time means the detection system can quickly respond to DDoS attacks. But if the session time is too small, the detection accuracy decreases.We

used F1 score and FPR as a performance measure for the third research question.

In order to answer the first research question, we have made visualization of feature importance on our dataset using box plot. The result showed that request rate and download rate have higher contribution for detection among all the five features. Furthermore, we used random forest classifier on our dataset to study the practical contribution of each feature. Random forest classifier assigns feature importance value for each feature during training. The result obtained agrees with our qualitative analysis.

In order to answer the second research question, we have tested Gaussian Naive Bays, Decision Tree, AdaBoost ensemble learning, Random forest and support vector machine supervised classifiers on our dataset.

We were able to distinguish between APP-DDoS attack and flash crowd using our proposed approach with high detection accuracy and low computational complexity. Decision tree, AdaBoost and Random forest classifiers had F1 score above 99%. But decision tree has low computational complexity. We have selected Decision tree as our supervised classifier for detection. With decision tree classifier, we have achieved 99.445% F1 score and 0.47% FPR.

In order to answer the third research question, we generated datasets for 20 seconds, 40 seconds, 1 minute,2 minutes, 3 minutes,4 minutes, 5 minutes, 6 minutes,7 minutes and 8 minutes session time. We used one classifier selected in research question two, decision tree, and tested the classifier on the above datasets.

The highest F1 score of 99.625% was achieved on the 120 second dataset and lowest F1 score of 99.355% on 40 second dataset. Session time has direct impact on the response time of our proposed approach.

The smallest session time we have is 20 second. We compared the F1 score and FPR of 20 second session time to 120 second session time. The difference is very small. The advantage of using higher session time is very small in terms of detection accuracy. But using higher session time means the detection system takes higher response time. For the aforementioned reasons, 20 second session time is recommended to be used during feature computation.

1.1 Problem statement

Most researches done on application layer DDoS attack detection do not consider the similarity between legitimate flash crowds and application layer DDoS attacks. As a result, the proposed approach usually fail during flash events. Attackers take this as an advantage and try to hide their attacks behind flash crowds to minimize detection. Due to the similarity between attacks and flash crowds, the detection system may classify legitimate user as attack incorrectly. This may create problem for websites because it may lead to losing customers.

Separating flash crowds and DDoS attack is very important because the response is different for each event. To handle flash crowds, the correct measure is increasing server resources and bandwidth. But doing the same in case of attacks is inefficient usage of resources. The correct action should be terminating connection of attack requests. The research problem addressed in this research is **separation of application layer DDoS attacks from legitimate flash crowds**.

1.2 Objective

The general objective of this research is to investigate how to distinguish application layer DDoS attacks from legitimate flash crowds using machine learning approach.

1.2.1 Specific objectives

- Study literature and identify important features that distinguish application layer DDoS attack from legitimate flash crowds
- Propose detection system that correctly identifies application layer DDoS attacks from flash crowds
- Evaluate the proposed detection system

1.3 Scope

This research deals with application layer DDoS attack that occur in the presence of flash crowds. Generally, DDoS attack detection stages are **input processing, detection and mitigation**. The research focuses only on input processing and detection stages. The mitigation stage will not be addressed in this research. The mitigation stage deals with the action taken based on the decision obtained from the detection stage.

This thesis does not deal with network layer DDoS attacks. Network layer DDoS attack exploit network and transport layer protocols such as TCP and UDP. In addition, this thesis does not deal with application layer DDoS attack that occur when there is no flash crowd.

1.4 Research methodology

In order to achieve the research objectives, we followed the following procedures.

- Literature review was done to understand the problem, study the strength and weakness of other existing approaches
- Features that help distinguish application layer DDoS attacks from legitimate flash crowds were identified from literature review
- A supervised machine learning based detection approach was proposed

- Existing machine learning algorithms were studied and few candidate classifiers were selected
- A dataset that contains examples from both application layer DDoS attack and legitimate flash crowds was prepared
- The selected candidate classifiers were evaluated in terms of detection performance as well as computational performance using the prepared dataset
- The best classifier was selected and recommended for application layer DDoS attack detection system by analyzing the evaluation result of candidate classifiers

1.5 Thesis outline

Chapter 2 provides background information on the nature of application layer DDoS attack and briefly describes HTTP protocol. Chapter 3 discusses the work of related researches in the area of application layer DDoS attack detection. Chapter 4 discusses our proposed approach. Chapter 5 discusses the evaluation case study of the proposed approach. Chapter 6 presents result and discussion of the evaluation. Finally, chapter 7 provides conclusion and recommendation.

Chapter 2

Background

In this chapter, we provide background information on application layer DDoS attack and flash crowds. We begin with introduction of Hyper text transfer protocol (HTTP). In this research, we are concerned with HTTP application layer protocol. Both flash crowd and application layer DDoS attack use HTTP protocol to communicate with server.

Introduction to distributed denial of service attack (DDoS) will also be discussed. Types of application layer DDoS attacks and their description are provided. The last sub section deals with flash crowds.

2.1 Hyper text transfer protocol(HTTP)

HTTP (Fielding et al., 1999) is an asymmetric request-response client-server protocol. HTTP protocol is implemented by both server and client. By asymmetric we mean that the client pulls information from server by sending request but not the other way. HTTP works over TCP/IP protocol. HTTP protocol is very popular application layer protocol used in the web (Maier et al., 2009). HTTP is not limited to transfer of hypertext. HTTP is also used in distributed, collaborative and hypermedia information systems.

In HTTP protocol, the client sends a request message to the HTTP server. Then the server sends a reply message called HTTP response. One example of HTTP client is a web browser. When we go to certain website using our web browser, the web server prepares an HTTP request using the website URL. The server interprets HTTP request from the browser and returns an HTTP response containing the requested web object if available.

Features of HTTP protocol (Fielding et al., 1999)

- HTTP is stateless protocol which means that current request is independent of previous request and it is not possible to exchange information between subsequent requests.
- HTTP allows negotiation of data type and representation which allows application that run on different platforms to interact easily with each other
- HTTP is connectionless protocol meaning that after the client sends a request, it can terminate the TCP connection. Then the server reconnects to the client and sends the response.
- HTTP is data independent meaning that any type of data can be sent by HTTP as long as the client and server agree on the data type.

HTTP protocol parameters

There are many parameters of HTTP protocol we will discuss some parameters.

- HTTP version: a string indicating the HTTP version such as HTTP/1.0 or HTTP/1.1.
- Uniform resource identifier (URI): is a formatted case-insensitive string containing name and location that uniquely identifies a resource in Internet. The resource can be website, web service and so on. URI contains protocol, the domain name or IP address, port and finally the web object. The default HTTP port is port 80. http::/www.aait.edu.et is one example of URI
- Date/Time stamp: represented in Greenwich Meridian Time (GMT).
- Character set: states the character set preferred by the client
- Media type: contains the file type of the media.
- Language Tag: is used to indicate the language used by the client

Contents of HTTP message

- Message Header: Header Fields provide meta data information about the request. There are four types of HTTP headers: General-header, Request-header, Response-header and Entity-header. General-header is used for both request and response. Request-header, response header and Entity-header which contain information about the requested resource.
- Message body: Contains the actual HTTP request data such as forms and uploaded files from client. And, the response data on server

HTTP request methods

HTTP GET request is usually used when the client retrieves data. HEAD is another request method in which only header section and status are transferred. POST: post request is used when client wants to send data to the server.

Components of HTTP response

Status code: It is a three digit code that indicates the status of a request. Table 2.1 shows the meaning of some HTTP status codes.

Table 2.1: HTTP status codes and their meaning

status code	Meaning
1xx	A status code that starts with 1 indicates the request is received
2xx	The request is successfully received and understood by the server. For example, 200 means the request is successful
3xx	Indicates redirection to another page
4xx	Indicates that there is some error in the client. For example, 404 indicates the requested resource is not found
5xx	Indicates that there is some error in the server. For example, 503 means the requested service is not available

2.2 Application layer DDoS attack

Denial of service (DoS) attack is an attack against availability of service. DoS attacks are characterized by their attempt to prevent legitimate users from accessing the target website by deliberately sending large number of requests in order to quickly deplete target resource. Resources can be bandwidth, memory or processing time. When the target is DoS attacked, it will not be able to serve legitimate user requests creating denial of service.

The potential targets include but not limited to websites, web services, information communication technology based infrastructures, power grids and weapon systems (Kumar et al., 2009). The consequences of such attacks can be devastating.

When a single source is used to perform DoS attack, the attack will be ineffective and can be easily detected. When DoS attack is generated from multiple geographically distributed machines it is called Distributed Denial of Service (DDoS).

In order to perform DDoS attack, attackers assemble a large number of machines called Zombies (Alomari et al., 2012). Group of such Zombie machines is called a bot-net. The zombie machines are either compromised by attackers or intentionally used for DDoS attacks. In most cases, machines with weak network security are compromised by attackers. The owners may not even know their machine is participating in DDoS attack.

DDoS attack architecture

Commonly used DDoS attack architecture models are agent handler model, Internet Relay Chat(IRC) model and reflector model (Specht and Lee, 2004). The models are not mutually exclusive a DDoS attack may use combination of the architecture models.

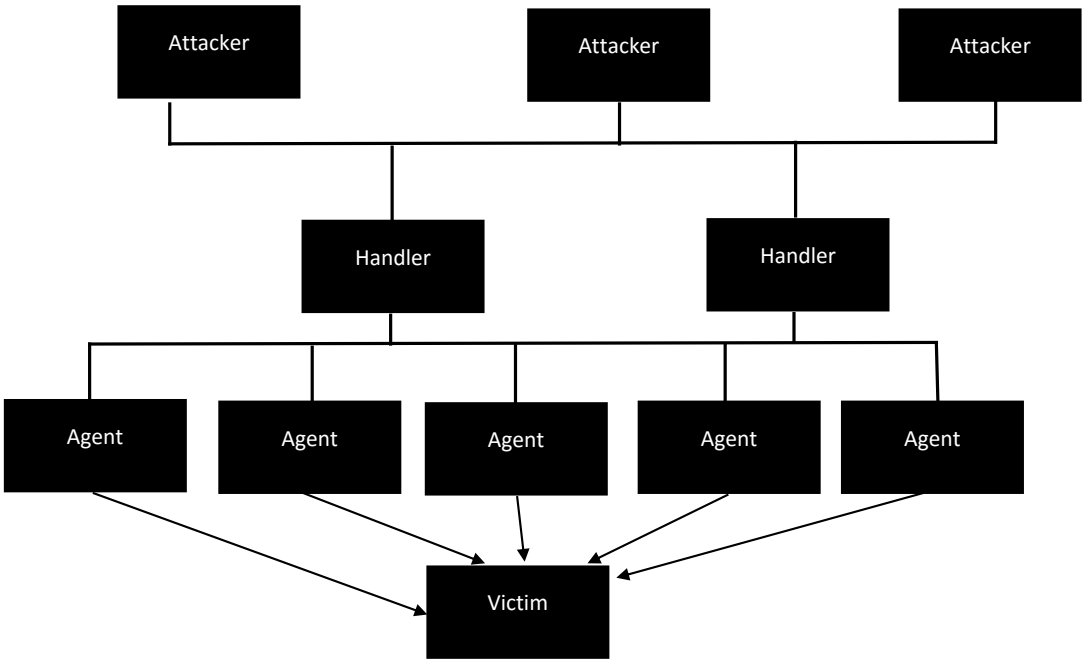


Figure 2.1: Agent Handler model

Agent handler model has client, handler and agent entity. Client entity represents the machine that is used by the attacker to communicate with other entities. Handlers are programs distributed over the Internet. Attackers use handler to communicate with agent entity. Handler's mange group of agent entities. Agents are machines that conduct the DDoS attack.

Reflector model is a modified version of agent handler model. In this model, agents are instructed to send streams of request packets with the victim's IP address as a source to reflectors. Any machine that responds to a request can be a reflector. Web server, DNS server and FTP server are examples of reflectors. The reply that come from the reflectors creates a denial of service attack on the victim's machine.

In IRC based model, attacker uses IRC channels instead of handlers to directly access agents or zombie machines. IRC servers have high traffic so it is difficult to intercept communication between attacker and zombie. Attackers share list of zombie machines to create a large army using IRC services.

There are many free tools to conduct DoS attacks. Some tools can even be used by anyone without expert knowledge. Some popular DoS attack tools are Low orbit Ion Cannon (LOIC), High Orbit Ion Cannon (HOIC), SlowLoris, R U DEAD Yet(RUDY) (Specht and Lee, 2004).

DDoS attacks can target any data communication layer. But most common DDoS attacks target either network layer or application layer. Network layer DDoS attack utilize network layer protocols such as IP and ICMP and transport layer protocols UDP and TCP. Network layer DDoS attacks are out of the scope of this research.

Application layer DDoS attacks

Application layer DDoS attacks target application layer protocols such as HTTP, FTP and SMTP. In order to classify a DDoS attack as application layer attack, the attacker has to setup a legitimate TCP connection to the target. Application layer DDoS attacks require less resources compared to network layer DDoS attacks. Application layer DDoS attack can disrupt the function of web based applications such as searching, financial transaction and other database operations.

For DDoS attack to be successful, the attacker has to send more requests than the target server can handle. the efficiency of DDoS attack can be measured in terms of the ratio of the resources used to process the request divided by the resources required by attacker to send request. The resource can be bandwidth, memory or processing time (Xie and Yu, 2009). Application layer DDoS attacks have higher amplification effect. The reason for this is that a single HTTP request may require request processing, database transaction, loading files from disk and preparing response.

(Ranjan et al., 2006) classified application layer DDoS attacks in to three. ***request flooding attack, asymmetric workload attack and repeated one shot attack***.

Request flooding attack is conducted by sending HTTP requests in higher rate compared to legitimate clients. Request flooding attack works by repeatedly sending get request to web pages of the target web server. Request flooding attack tools usually use a large pool of IP address to avoid detection. Each request is performed by setting up a full TCP connection with a server with valid IP address.

In asymmetric workload attack, attacker uses requests that require heavy processing on the server side. Requests that require database access such as search queries are often used. The objective is to overwhelm the server with few requests that require higher workload from the server. The advantage of such types of attack is that with small attacker resources it is easy to take down a target server. Prior knowledge about the server workload should be known by attackers.

Repeated one shot attack is a combination of request flooding and asymmetric attack. In this case, the attacker sends one asymmetric request per one session. It is very difficult to identify such type of attacks because the request rate per session resembles legitimate users' activity.

2.3 Flash crowd

Flash crowd is defined as a sudden, large surge in traffic to a particular website. It is characterized by an exponential rise in number of requests above average. Large number of legitimate users request that information from a website during flash crowd.

Flash crowds may happen when breaking news is added to a website, when a new popular product or service is released and when a strange event is posted on a website. Announcement of IOS 7 by apple on September 18, 2013, Sandy Storm reported on October 31, 2012 in USA, Death of Michael Jackson from June 24 to June 31 and world cup 98 from May 1 to July 24 1998 are popular examples of events that caused flash crowd (Bhandari et al., 2016).

(Bhandari et al., 2016) classified flash crowds based on *predictability, duration, shock level, geographical distribution and traffic type.* Flash crowds can be predictable or unpredictable. When a website is aware of the possibility of flash crowd the flash crowd is called predictable. For example, when a new product is released, the website that hosts the product should expect a flash crowd. When the website server is unaware of the coming flash crowd, it is called unpredictable flash crowd. For example, when some popular event such as the death of famous person happens, popular search engines may suffer from a flash crowd. When a website is linked by the website where flash crowd happens, the linked website may be victim of unpredictable flash crowd.

The flash crowd time may range from few hours to days based on the type of the event. For example, world cup 98 caused a flash crowd for the whole duration of the month. In other case events that last for small duration such as sport games may cause a short

duration flash crowd.

Flash crowd can also be categorized using shock level. Shock level measures the increase in flash crowd requests. Shock level is related to the popularity of the flash event and popularity of the website that reported the event.

The geographical distribution of flash crowds may be local, regional or global. Geographical distribution is determined by the location of the flash events and location of interested followers. For example, a news that only concerns one nation will most probably create a regional flash crowd. Whereas an international news may cause global flash crowd.

When the flash crowd is directly generated on a website it is called direct flash crowd. For example, when people search about a popular event, the flash crowd is directed on search engines. When a popular website links to some small website, the small website will suffer from indirect flash crowd. This effect is known as slash dot effect (Bhandari et al., 2016).

In this chapter we have provided background information about DDoS attacks, APP-DDoS attacks and flash crowds. APP-DDoS attacks are classified as request flooding, asymmetric and repeated one shot attacks. Flash crowd is a sudden surge in traffic related to the addition of popular content to a website. DDoS attacks and flash crowds have similar effect of denial of service but the intention of DDoS attack is malicious whereas flash crowds are legitimate requests.

Chapter 3

Literature review

Distributed Denial of service (DDoS) attacks target victims' server by deliberately holding resources and denying service to legitimate users. Denial of service attacks can target any OSI layer. The most common attacks are network layer attacks and application layer attacks. We classified related literatures in to two: those that address network layer DDoS attacks and address application layer DDoS attacks. We further classified literatures that focus on application layer DDoS attack detection based on detection against normal activity and detection against flash crowds.

3.1 Network layer DDoS attack detection

There are many literatures that cover DDoS attacks that rely on flaws of network and transport layer protocols. Quality of service regulation by using rate control and window control was proposed to detect DDoS flows from normal flows that operate on nearby routers to the victims' server (Garg and Reddy, 2001). Another approach uses Total variation and Bhattacharyya coefficient probability metrics to distinguish between DDoS, normal and flash crowds (Li et al., 2009), a hybrid of neural network and fuzzy logic based DDoS detection approach was proposed in the work of (Fitsum, 2008).

The problem of distinguishing between DDoS attack and flash crowds using network layer information was discussed in the works of (Kandula et al., 2005; Sachdeva et al., 2016; Yu et al., 2009, 2012). (Kandula et al., 2005) proposed graphical tests to quickly identify DDoS attacks from flash crowds. The graphical test is provided during first page request. No TCP connection is allowed before passing the test. Another approach to discriminate DDoS attack from flash crowds is to compare the packet flow on the network. The assumption behind this is that DDoS attack flow have more similarity compared to flash crowd flow. In order to discriminate between the flows of DDoS and flash crowd, flow

correlation (Yu et al., 2009) and information distance (Yu et al., 2012) are suggested. (Sachdeva et al., 2016) used IP address entropy and IP cluster entropy to discriminate between DDoS and flash crowds.

3.2 Application layer DDoS attack (APP-DDoS) detection

In this section, we will present review of literatures that focus on application layer DDoS (APP-DDoS) attack detection. We categorized the literatures in to two: APP-DDoS attack detection against normal activity and application layer DDoS attack detection approach against flash crowds.

3.2.1 App-DDoS attack detection against normal activity

When the DDoS attack targets application layer it is called application layer DDoS (APP-DDoS) attack. APP-DDoS attack require the attacker to setup a legitimate TCP or UDP connection. Most solutions to detect at the network layer may not detect APP-DDoS attacks. Information obtained from application layer protocols such as HTTP is required for effective DDoS detection. There are few researches that cover the problem of discrimination between APP-DDoS and normal activity (Xie and Yu, 2006; Ranjan et al., 2006; Wang et al., 2010; Patil and Kulkarni, 2011; Xu et al., 2014; Yadav and Selvakumar, 2015).

(Ranjan et al., 2006) classified APP-DDoS attacks in to request flooding attack, asymmetric workload attack and repeated one shot attack. They also tried to investigate the impacts of APP-DDoS attack. In the same research, a continuous suspicion assignment was suggested. The suspicion was computed by first obtaining a distribution of legitimate user activity. The legitimate distribution was computed by considering features such as session arrival rate, session request rate and request workload. During detection, a suspicion value is assigned by comparing the request profile with the legitimate distribution. The suspicion assignment is considered for scheduling requests on the proxy server.

Another detection approach assumes that page visiting or access behavior of normal activity is different from DDoS attack. (Xie and Yu, 2006; Xu et al., 2014) tried to model the legitimate user access behavior as a transition between one page to the other. Page transition was modeled using Hidden Semi-Markov model on the work of (Xie and Yu, 2006) and using random walk graph on the work of (Xu et al., 2014). A deviation from the legitimate distribution is considered as APP-DDoS attack. The problem with this approach is that attackers may program their zombies to follow a legitimate access pattern by analyzing the content of the target website.

(Patil and Kulkarni, 2011) proposed assignment of trust value computed based on users visiting history. Request inter-arrival time is considered for computation of trust value. Trust value is stored on the client as a cookie and appended with every client request. The proposed scheduler schedules incoming request based on their trust value. This approach is vulnerable to APP-DDoS attacks with IP address spoofing.

(Beitollahi and Deconinck, 2014) used statistical analysis using users browsing history to assign a score value to users. Features such as request rate, download rate, request arrival rate and page popularity were used to assign scores. This approach favors regular website clients against new clients.

The popularity of web pages can also be used to distinguish between APP-DDoS attack and normal activity. The assumption behind this is that attackers do not care which particular page is requested. But normal users care about the content of the requested pages. (Wang et al., 2010) suggested a detection method using relative entropy. The detection method first uses legitimate requests to cluster or group web pages based on their click ratio. Click ratio is defined as the ratio of number of requests of a particular page to the sum of requests of all pages in the website. The relative entropy of the incoming session is compared to all clusters during detection. When attackers target either the home page or most popular pages, it is difficult to detect APP-DDoS attacks using this approach.

(Yadav and Selvakumar, 2015) considered features that are directly available on server logs and additional derived features for detection. Principal component analysis (PCA) was applied to reduce the number of features. Logistic regression supervised machine learning algorithm was used for detection. They obtained 98.64% detection rate and 1.41% FPR. The result is good but their approach should be tested with flash crowds.

3.2.2 APP-DDoS attack detection against flash crowd

Due to the close resemblance between APP-DDoS attacks, the case of flash crowds may be considered when designing APP-DDoS attack detection (Bhandari et al., 2016). A taxonomy of flash crowds and some features that help differentiate APP-DDoS attacks from flash crowds were discussed in the work of (Bhandari et al., 2016). Features such as distribution of requests among source IP, geographical distribution of source IP, URL access behavior and change in rate of request were suggested (Bhandari et al., 2016). They used world cup 98 dataset to model flash crowds and created APP-DDoS attack using simulation to investigate the significance of the suggested features.

Page popularity and page access transition were suggested to identify APP-DDoS attack from legitimate flash crowds by (Yu et al., 2007) and (Xie and Yu, 2009; Ye and Zheng, 2011) respectively. (Yu et al., 2007) suggested page access entropy by assuming that the entropy of flash crowd page access is different from APP-DDoS attack. This approach may not work when the attacker requests popular pages by studying the website. (Ye and Zheng, 2011) proposed the transition between web pages for detection. (Xie and Yu, 2009) modeled spatial and temporal user access patterns of flash crowds using hidden Semi-Markov model.

Another approach that uses a combination of network layer and application layer features was suggested in (Ramamoorthi et al., 2011). It uses features such as HTTP request rate and page viewing time from application layer and session rate, number of TCP, UDP and ICMP packets from network layer. Enhanced support vector machine with string kernel was used to model legitimate flash crowd. They obtained a classification accuracy

of 99.32%.

Request rate, page popularity and page access pattern were commonly used features for detection of APP-DDoS attacks against normal or flash crowd. Using one or two features alone is not enough for robust APP-DDoS detection. For example, detection systems that rely on page popularity may fail when the attacker studies the website to identify most popular pages and then programs its zombies to request most popular pages. Again, if the detection system considers page access transition for detection, the attacker may easily program its zombies to follow a similar access pattern to that of legitimate users. This shows that using a combination of the above features will make the detection system more robust.

In this research, we propose APP-DDoS detection approach that uses a combination of features that were suggested by related researches. We further investigate the relevance of those features for the detection of APP-DDoS attack against legitimate flash crowds. We also investigate if we can successfully detect APP-DDoS attack that occur together with flash crowd using a combination of suggested features.

Chapter 4

App-DDoS attack detection approach

We propose an application layer DDoS attack detection system that has two stages. The stages are **feature computation from server log and detection stage based on the computed features**. The input for feature extraction stage is web server log data. Web server logs contain information about the requests made by clients. Server log information includes the client address, time stump, URL of the requested object, reply size and client browser information. It is difficult to have accurate attack detection by considering only the information available on server logs. Some literatures suggested additional features that are derived from basic server log information (Xie and Yu, 2006; Ranjan et al., 2006; Wang et al., 2010; Patil and Kulkarni, 2011; Xu et al., 2014; Yadav and Selvakumar, 2015). We have selected *Request rate, page popularity, request inter-arrival time, download rate and ratio of successful requests* (Beitollahi and Deconinck, 2014; Yadav and Selvakumar, 2015). All features suggested in related literatures were considered during selection. The selection was done by analyzing potential contribution of the feature for detection of DDoS attack and the computational requirement of feature in terms of memory and processing time.

All features are computed for each unique client by considering a predefined time interval called session time. Client is defined as the source of the request identified by IP address. Each client has its own unique IP address. The computation of these features will be discussed in Section 4.1.

Application layer DDoS attack detection in the presence of flash crowds is a binary classification problem. The input of the detection stage is the value of features extracted in the feature extraction stage. The expected output of the detection system is either the client is legitimate or attack. In the detection stage, we put a supervised learning classifier to make a decision.

The mitigation stage could use information obtained from the detection stage to block any pending current and future APP-DDoS requests. The IP address of the attack client will be added to a blacklist. *Mitigation stage is not the focus of this research.* Figure 4.1 shows the stages of the proposed approach.

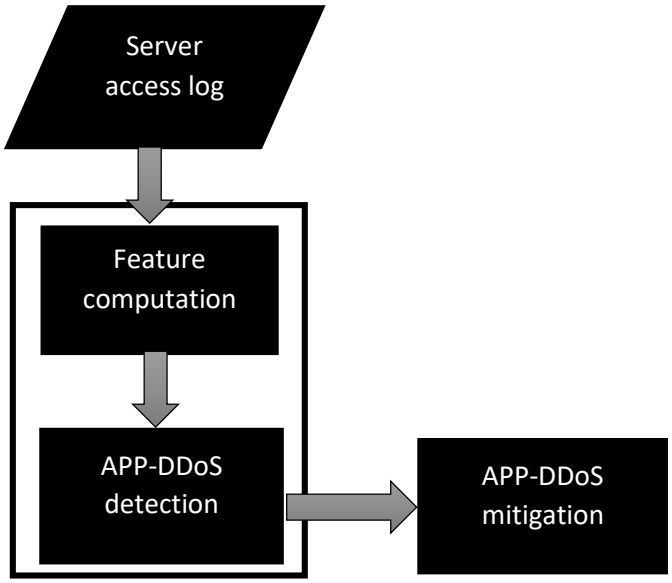


Figure 4.1: App-DDoS attack detection and mitigation stage

4.1 Feature computation

4.1.1 Server access log

Web servers register basic information about each request such as **Request address, time stamp, URL, request type, response code, replay size and user agent information**. Each entry in a web access log represents one request. One example entry of apache web server access log is shown below. The URL is relative to the web server's home directory. The time stamp has one second precision.

::1 - - [17/Nov/2016:21:02:52 +0600] "GET /PhpProject1/index.php HTTP/1.1" 200 2109 "-" "Mozilla/5.0 (Windows NT 10.0; WOW64; rv:39.0) Gecko/20100101 Firefox/39.0"

4.1.2 Feature computation

The features used in the detection of App-DDoS attack from flash crowds are **Request rate, page popularity, download rate, request inter-arrival time and ratio of successful requests to total requests**. The description of the features is provided in the following paragraphs. All features are computed using a predefined **session time** (T). **Session time** (T) is a time interval in which all requests that arrive in that interval are considered together when computing features.

Request rate (RR) is defined as the number of requests that arrive in a session time divided by session time. RR is computed for each unique client identified by its IP address. Request rate can be derived from server logs by counting the number of requests served in T for each unique client. Equation 4.1 is used to compute request rate.

$$RR = \frac{N_r}{T} \tag{4.1}$$

where N_r is number of requests and T is the session time.

Request rate is chosen because it helps detect request flooding attacks. In request flooding attack, attackers send higher number of requests compared to genuine users.

Page popularity (PP) Page popularity is defined as the number of requests of an object divided by total number of requests of all objects in the same website. A web object includes web pages, image, audio, video, script files, style sheet files and any other components of a website.

Before using page popularity for detection, we need to compute average popularity value of all web objects in a website. To compute average popularity, we will consider server log collected during normal operation of the website. From the collected server log, we compute the page popularity value of each web object using Equation 4.2. After this we will obtain a page popularity value corresponding to each web object.

$$PP_j = \frac{NR_j}{NR_t} \tag{4.2}$$

where PP_j page popularity of object j, NR_j the number of requests of object j and NR_t total number of requests of all objects in the considered period.

In the detection stage, we will take the average page popularity value of each web object requested by the client in a session time. The average popularity value of each requested object in the session time (T) is summed up using Equation 4.3

$$PP = \sum_j NR_j * PP_j \tag{4.3}$$

where NR_j the number of requests of object j and PP_j page popularity of object j. All requested pages in the window time are considered in the summation. PP is computed for each client.

Download rate (DR) Download rate is defined as total number of bytes of reply of all requested objects in a session time divided by session time (T). When the requested object is not found on the server, the reply size is taken as zero. Equation 5.1 is used to compute download rate of each client.

$$DR = \frac{\sum_i Reply_i}{T} \tag{4.4}$$

Where DR is download rate and $Replay_i$ the replay size in bytes for request i and T is session time.

Request inter-arrival time (RIA) Request inter-arrival is defined as the time duration between current request and previous request. The inter-arrival time between all requests in a session time are summed up. Equation 4.5 is used to compute request inter-arrival time of each client in a session time (T).

$$RIA = \sum_{k} (t_k - t_{k-1}) \tag{4.5}$$

Where t_k is time stamp of request k and t_{k-1} time stamp of the predecessor request $k-1$

Ratio of successful request to total requests (RSR) Requests with a reply code of 200 are considered as successful requests. RSR is defined as the ratio of requests with a reply code of 200 divided by total number of requests in session time. RSR is computed for each unique client. Equation 4.6 is used to compute RSR. The value of RSR is between 0 and 1.

$$RSR = \frac{NRP}{NRT} \tag{4.6}$$

Where NRP is the total number of requests with 200 reply code that occur in session time (T) and NRT total number of requests in the session time (T) Table 4.1 discusses why we select the above described features to distinguish APP-DDoS attack from flash crowd.

Table 4.1: Reason of selection of features

Feature	Selection reason
RR	Request flooding APP-DDoS attack is characterized by high number of requests per client whereas the number of requests per client is small for flash crowd. RR is selected for the detection of Request flooding attack
PP	Legitimate users in a flash crowd tend to access popular pages more frequently because they look for similar news. But APP-DDoS attacks requests pages randomly. The PP value of APP-DDoS attack is lower than PP value of flash crowd
RIA	Normal users take some viewing time before requesting the next object. Whereas, APP-DDoS attack is generated by machines that do not need viewing time. So the request inter-arrival time is smaller for APP-DDoS attacks compared with legitimate users in a flash crowd
DR	When a page is requested to the server, a disk access operation is performed. The disc access time depends on the size of the requested object. Large size web objects require higher disk access time. Large size Web objects can be selected to conduct Asymmetric APP-DDoS attacks. But normal users do not intentionally request only large size web objects. This creates a difference in download rate between APP-DDoS and flash crowd
RSR	APP-DDoS attackers may request web objects that does not exist in the server. This makes the sever to reply with 404 error message. But usually, legitimate users in flash crowd has very low probability of requesting an object that does not exist in the website. This create a difference in RSR between flash crowds and APP-DDoS attack

4.1.3 Feature scaling

The values of each feature used in the detection system has different range. For example, the download rate is usually in the range of thousands while others are in the range of decimal fractions. Some classifiers such as decision tree and Adaboost does not require all the features to be in similar scale while Support Vector Machine requires all inputs to be on the same range (Graf and Borer, 2001).

We can apply feature scaling in order to make the values of all features in similar range. In order to transform a feature distribution to a normal distribution with a mean of zero and unit standard deviation, we can use Equation 4.7

$$x_n = \frac{x - \bar{x}}{\sigma} \tag{4.7}$$

Where x_n is the transformed feature value, x is original feature value, $\bar{x}$ is the mean of all feature values and σ is the standard deviation of all feature values.

4.2 Detection stage

The input of the detection stage is an array of five feature values corresponding to RR, PP, DR, RIA and RSR respectively. The output of the detection system is either one or zero. One means the input feature vector corresponds to APP-DDoS while zero means the input feature vector corresponds to flash crowd.

4.2.1 Supervised machine learning classifiers proposed for detection

Our detection stage uses a supervised machine learning classifier. Our problem is a binary classification problem where the output is either positive or negative.

There are a number of candidate supervised learning classifiers. From them we have chosen Gaussian Naive Bays classifier, decision tree classifier, random forest ensemble classifier, AdaBoost ensemble classifier and support vector machine classifier. Those classifiers are chosen because they represent most types of supervised learning approaches. The brief description of the candidate classifiers is discussed below.

Gaussian Naive Bays classifier

Naive Bayes methods are a set of supervised learning algorithms based on applying Bayes' theorem. **The assumption of Naive bays classifiers is that every pair of features**

are independent. If we assume that all features are independent with each other the Equation 4.8 holds

$$P(x_1,...,x_N|y) = \prod_{i=1}^{N} P(x_i|y) \tag{4.8}$$

Where y is output class value and N is number of features

Given a classification variable y and independent feature vector $x_1,x_2....x_n$, Bayes' theorem states the following relationship:

$$P(y|x_1,....,x_n) = \frac{P(y)P(x_1,....,x_n|y)}{P(x_1,...,x_n)} \tag{4.9}$$

Combining 4.8 and 4.9 we get

$$P(y|x_1,....,x_n) = \frac{P(y)\prod_{i=1}^{N} P(x_i|y)}{P(x_1,...,x_n)} \tag{4.10}$$

The denominator $P(x_1,...,x_n)$ is constant value given the input.

We can use the above equations to generate the following classification rule

$$y = argmax_y P(y) \prod_{i=1}^{N} P(x_i|y) \tag{4.11}$$

$P(y)$ is the relative frequency of class y in a training data. $argmax_y$ is interpreted as y such that $P(y)\prod_{i=1}^{N} P(x_i|y)$ is maximum.

The difference between the naive Bayes classifiers comes on the assumption of $P(x_i|y)$. Gaussian naive Bayes assume Gaussian distribution for $P(x_i|y)$

$$P(x_i|y) = \frac{1}{\sigma_y\sqrt{2\pi}} e^{-(x_i-\mu_y)^2/2\sigma_y^2} \tag{4.12}$$

σ_y and μ_y are estimated using maximum likelihood estimation.

Even though the assumption of naive Bayes classifier usually does not hold, naive Bayes classifier have good performance on document classification and Spam filtering problems (Ting et al., 2011; Metsis et al., 2006). They require a small amount of training data to estimate the necessary parameters.

Support vector machine(SVM) classifier

Support vector machine is one of the popular supervised learning algorithms. SVM works by creating a maximum margin separator which is a decision boundary with the largest possible distance between points in different class. For example, for a binary classification problem SVM works by searching a hyperplane that separates the positive examples from negative example.

Data points that are closest to the decision boundary are called **support vectors**. Support vectors determine the selected decision boundary. Any data point that is not support vector does not affect the decision boundary. The main intention of SVM is to minimize generalization error by creating a decision boundary farthest from the seen training examples.

SVM can be applied for linearly separable problems very easily. But it can also be extended to deal with non-linearly separable problems using kernel trick. The kernels are used to separate a non-linearly separable problem by transforming the inputs in to higher dimensional space.

If the training data is not linearly separable, we can use kernels that map the input space to a higher dimensional space. There are many kernels available such as Linear, Polynomial, radial basis function kernel and sigmoid kernel. The choice of kernel is based on the nature of the dataset.

Decision tree classifier

Decision tree is another popular machine learning algorithm that is used for classification and regression problems. A decision tree has branch nodes and leaf node. Each branch node represents a choice between alternatives. Each leaf node represents a decision.

Decision tree reaches its decision by performing a series of tests starting from the root and following a path to the leaves. A branch node corresponds to a test on one feature value. The paths from the node will have all the possible values of the node feature. By

traversing a decision tree starting from the node we will obtain the final answer when we reach leaves.

Like any other supervised machine learning algorithms, the learning instance is represented by attribute value pairs. The attribute value pairs can be continuous or discrete.

The advantages of decision tree classifier:

- It is natural decision-making process so it is understandable by humans. It is also easy to visualize how the decision tree model classifies an instance.
- It is computationally efficient. It uses a greedy search over the feature space. Decision tree has implicit feature selection by measuring feature importance and selecting the most important feature as a node on every iteration during training.
- Decision trees require relatively little effort of data preparation. They can easily handle missing feature value and do not require scaling or normalization

The disadvantage of decision trees is that they need proper pruning. Pruning is the process of reducing the size of decision trees by removing sections of the tree that contribute little to make decision. Pruning reduces the complexity of the final classifier and over fitting.

ID3(Iterative Dichotomiser 3) decision tree training algorithm

There are many decision tree types such as ID3, C4.5 and CART (classification and regression tree). However, we will discuss ID3 decision tree training algorithm because most decision tree implementations use a variant of this algorithm. ID3 learning algorithm was developed by Ross Quinlan (1983). ID3 learning algorithm works by performing a greedy search to test attribute at every node. In order to identify the most useful feature, it uses information gain metric. The feature with the highest information gain is selected as a node. This process continues recursively by considering features that are not used before and calculating the information gain.

The algorithm stops when one of the following conditions happen

- When every element in a subset belongs to the same class. In this case the node will become a leaf.
- When there are no more features to be selected but the examples in a set belong to more than one class. In this case the node is turned in to a leaf with the most common class of examples in the subset
- When there are no examples in the parent set that match the value of the node attribute. In this case the node will be a leaf and take the most common class of the parent set.

When a set S is split by attribute A in to T subsets, the information gain is given by Equation 4.13

$$Gain(A,S) = Entropy(S) - \sum_{t=1}^{T} p(t) * Entropy(t) \tag{4.13}$$

Where $p(t)$ is the proportion of the number elements of subset t to the number of elements of set S. Entropy is calculated using equation 4.14

$$Entropy(S) = \sum_{x=1}^{X} p(x) * log_2^{\frac{1}{p(x)}} \tag{4.14}$$

Where S is a set in which entropy is calculated and X is set of classes in S and $p(x)$ is the ratio of number of elements in class x to the number of elements in set S

Random forest ensemble classifier

Random forest is an ensemble learning algorithm that combines multiple decision trees on the hope that the combined performance is better than the individual decision of decision trees. The base learner in random forest is decision tree.

During training and tree construction, the split is done by selecting the best feature from a random set of features in contrast to normal decision tree learning. The purpose of this is to add randomness to features selection and to reduce the correlation between decision trees. If there is high correlation between the individual decision trees, then the average result will not be better than the individual results.

During classification when an input is entered into the system, it is run on all trees. Then the result is the weighted average of all reached terminal nodes. But fore categorical output class majority voting can be used.

The advantages of random forest algorithm are similar to decision tree algorithm. The run time of random forest algorithm is relatively fast, it does not require data normalization and handles unbalanced data.

AdaBoost ensemble classifier

Ensemble learning is machine learning technique that combines multiple hypothesis to make decision. Multiple hypothesis is constructed by using different distributions of the same training data. The reason for using different distribution of training data is that every hypothesis has different classification accuracy on certain training instances compared to others. It is possible to minimize the training data set error to zero by choosing enough number of hypothesis specialized on certain parts of the data. The combined hypothesis is obtained by simple majority voting.

Weak or base classifier is a classifier which has accuracy as low as little more than random guessing. Ensemble learning is very effective on week classifier compared to strong classifier. Decision stumps, logistic regression and Naive bays classifier can be used as base classifier.

The most popular ensemble learning techniques are boosting and bagging. bootstrap aggregating (bagging) works by randomly drawing multiple subsets of the training data and train different classifier. Then decision of each classifier is combined by a simple

majority vote.

Boosting algorithm works by repeatedly running a week classifier on various distributions of the training data and then combines the decision of each classifier. The main difference with bagging is that the training examples are chosen systematically in such a way that each classifier will be better on unique feature space and very high accuracy is achieved by combining a number of weak classifiers.

Decision stump as weak classifier

AdaBoost algorithm can use any weak classifier as a learner. we have chosen decision stamp as a base classifier. **Decision stump** is a one level decision tree. It has one internal node that is immediately connected to leaves. one feature is chosen to classify the training set in to two or more possible values. For a feature with continuous value, some threshold value is chosen to classify training examples. For feature with finite discrete values, decision stump contains leaf corresponding to each possible value.

For binary classification problem, the decision is either positive or negative. The selected feature value is used to classify the training data in to two groups, positive or negative.

AdaBoost algorithm operation

AdaBoost algorithm was introduced in the papers (Freund, 1990; Freund et al., 1996) AdaBoost produces the final hypothesis by combining the result of each individual hypothesis. Each training example is a pair of input feature vector and output value. The training set T is defined as

$$T = \{(x_1, y_1), (x_2, y_2), ...(x_n, y_n)\}$$

where each input x_i is a vector of input features and y_i is 1 or 0. The result of each hypothesis is either 1 or 0 for binary classification problem.

AdaBoost uses weighted training set. Each example in a weighted training set has weight ($w_i \geq 0$). The weight value is assigned from the same probability distribution which means that the sum of each weight value is one.

$$\sum_{i=1}^{N} w_i = 1 \tag{4.15}$$

Where N is number of training instances. The higher the weight value the higher the importance during learning.

Inputs to AdaBoost algorithm

The required inputs for AdaBoost algorithm are *labeled training data, Learning algorithm and number of hypothesis*. The learning algorithm is the weak or base classifier such as decision stump. The number of hypothesis is the number of weak learners that contribute to the final decision.

Initialization

AdaBoost algorithm first assigns equal weight value $1/N$ for each training instance where N is total number of training examples. On the first iteration, every training example is equally important. The training data is fed to the week classifier on the first iteration. The weight value for next iteration is updated based on whether the training instance is correctly classified or not.

Training

In the training phase, the weak learner is presented with all examples on each iteration, then error is computed for each training example. The training error in one iteration is the sum of all misclassified training example weights. The training error e_t for iteration t is given by

$$e_t = \sum_{i:h_t(x_i)\neq y_i} w_i(t) \tag{4.16}$$

where $h_t(x_i)$ is the classifier output for training example x_i and $w_i(t)$ is the weight of training example x_i at iteration t.

The weight value of each **correctly classified** training instance is modified using the following equation

$$w_{t+1}(i) = w_t(i) * e_t/(1-e_t) \tag{4.17}$$

For **incorrectly classified instances**, the weight modification is given by

$$w_{t+1}(i) = w_t(i) \tag{4.18}$$

The normalization factor Z_t is calculated as

$$Z_t = \sum_i W_{t+1}(i) \tag{4.19}$$

Each training weight is divided by Z_t to make the weight a probability distribution. Each hypothesis has its own weight value α calculated using the following equation

$$\alpha_t = 0.5 * \ln((1-e_t)/e_t) \tag{4.20}$$

The training is repeated until all hypothesis and their weight value are calculated. The final hypothesis $H_f(x_i)$ is the combined decision of each hypothesis $h_t(x_i)$ given by

$$H_f(x_i) = sgn(\sum_t \alpha_t * h_t(x_i)) \tag{4.21}$$

Training classifier

The training data is composed of input features and the corresponding label. The input feature x_i is a vector of dimension five i.e. Request rate, page popularity, download rate, request inter-arrival and ratio of successful request.

The output Y_i is a binary value that indicates weather the example represents DDoS attack or normal. Attack sessions will have a value of 1 while normal sessions will have a value of 0. We can train the classifier off-line.

Classification

After the classifier is trained it can be deployed at the detection stage to separate legitimate flash crowd from application layer DDoS attack. The output of the detection system is used as an input to the mitigation stage. The mitigation stage terminates current and pending requests of an attack client and adds the IP address to black list so that any future connection attempts are terminated. Any request from a legitimate client is processed as usual.

Our proposed approach is a supervised machine learning based approach to detect APP-DDoS attack from flash crowds. We have selected request rate, page popularity, download rate, request inter-arrival time and ratio of successful requests as features for detection. We have also presented six supervised classifiers that can be used for APP-DDoS detection.

Chapter 5

Evaluation

The problem this research tries to address is how to distinguish application layer DDoS attacks that occur together with legitimate user flash crowds. The proposed approach to detect App-DDoS attacks from flash crowds is explained in the previous chapter. In this research three research questions are addressed. The first research question deals with the contribution of each feature for detection. The second research question evaluates our APP-DDoS detection stage of our approach. The third research question deals with the effect of session time on App-DDoS attack detection.

5.1 Feature importance for detection

As discussed in the proposed approach we have selected *Request rate, page popularity, request inter-arrival time, download rate and ratio of successful requests* as features of detection. As a first research question, we are interested to investigate the actual contribution of each feature for detection.

In order to investigate contribution of features, we prepared a dataset containing examples from DDoS and flash crowd. The details of dataset preparation are discussed in section 5.4. To investigate the contribution of all features, we drew box plot for each feature using our dataset. Box plots are drawn as follows. First, we arrange the feature value from lower to higher. Then we divide the feature value in to four quartiles. Box is drawn to show values that lie in second and third quartile. The center of the box indicates the median of the feature value. To visualize the contribution of each feature, we plot box plots of both flash and APP-DDoS classes side by side for all features.

5.2 APP-DDoS detection

As a second research question, we investigate whether we can distinguish between APP-DDoS attack and flash crowds. The core part of our detection system is supervised classifier. The performance of the supervised classifier determines the performance of our detection system.

There are many supervised classifiers. It is very difficult to find a universal classifier that works on all types of applications. One classifier that is good for certain application may be bad for other application area. The proper approach is to try the data on a number of classifiers and choose the one that is best fitted for the specific application.

In this research Gaussian naive Radial basis, support vector machine, decision trees, random forest and AdaBoost ensemble classifier are chosen as a candidate supervised classifiers for the detection system. So we will use our dataset to compare the performance of each classifier.

Evaluation criteria

The output of the classifier in the proposed approach is binary. For binary classifiers, there are many performance measures available but related researches use *F1 score, false positive rate (FPR) and classification time* to evaluate their detection system. Hence, we will use the same performance measures in our work.

Classification time measures the time it takes for a classifier to classify a dataset. Classification time measures the computational complexity of classifier algorithm. In our proposed approach, we use off-line classifier training. Training time is not included when we measure classification time. Classification time is measured in seconds.

Before defining the evaluation metrics F1 score and FPR, we need to define some quantities related to binary classification as shown in Table 5.1.

Table 5.1: Definitions of events in binary classification

Name	Description
Positive	Positive or 1 represents an application layer DDoS attack instance in the dataset
Negative	Negative or 0 represents a legitimate flash crowd instance in the dataset
True Negative(TN)	Occurs when both the actual value and the predicted value are negative
True positive(TP)	Occurs when both the actual value and predicated value are positive
False Negative (FN)	Occurs when the predicated value is negative but the actual value is positive
False Positive (FP)	Occurs when the predicted value is positive whereas the actual value is negative

Detection rate (DR)

Detection rate is defined as the ratio of the number of correctly classified test instances divided by the number of total test instances. Detection rate is computed using equation 5.1.

$$DR = \frac{\sum_{i=1}^{N} y_t(i) = y_p(i)}{N} \tag{5.1}$$

Where y_t is the true value, y_p is predicted value by classifier and N is total number of test instances.

False positive rate (FPR)

False positive rate (FPR) is computed using equation 5.2

$$FPR = \frac{FP}{FP + TN} \tag{5.2}$$

Where FP is the number of false positives from the test data and TN is the number of true negatives from the test instances. The value of FPR is between 0 and 1 the lower

the value the better the performance of the classifier.

FPR is very important criterion because a classifier with high false positive rate will have higher probability of classifying a legitimate client as an attacker. Terminating legitimate client's session may annoy the client and discourage the client from using the website.

F1 Score

F1 score is the weighted average of precision and recall. Precision is inversely related to false positive rate while recall is inversely related to false negative rate. The definitions of precision and recall are discussed below. *F1 score* is defined in Equation 5.3

$$F1 = \frac{2 * precision * recall}{precision + recall} \tag{5.3}$$

Where *precision* is defined as the ratio of true positive to the sum of true positive and false positive.

$$precision = \frac{TP}{TP + FP} \tag{5.4}$$

Recall is defined as the ratio of true positive(TP) to the sum of true positive(TP) and false negative(FN)

$$recall = \frac{TP}{TP + FN} \tag{5.5}$$

The value of $F1$ score is between 0 and 1 and the higher the score the better the classifier.

F1 score is more appropriate than detection rate when the number of positive class and negative classes are not balanced. In most real-world problems, the number of negative examples are very high compared to positive examples. For instance, the legitimate flash crowd data is very large compared to DDoS data. Since there is no application layer DDoS data, the attack was generated using attack tool. Even the generated attack examples are very small compared to flash crowd. When positive and negative examples

are not balanced, detection rate leads to incorrect conclusion. When the number of positive and negative examples are balanced the F1 score and detection rate will have similar values. **F1 score is chosen together with False positive rate (FPR) as evaluation criteria for research question one**.

5.2.1 Investigation approach

In order to investigate APP-DDoS detection performance, we have followed the methodology shown in Figure 5.1. In this approach, first a dataset containing APP-DDoS attack and flash crowd examples is prepared. The dataset was prepared by selecting session time of 20 seconds. Then every candidate classifier was evaluated using the prepared dataset. 10-fold cross validation is used to evaluate all candidate classifiers. F1 score and FPR score is computed for each candidate classifier. The details of dataset preparation and classifier evaluation are discussed in Section 5.4.

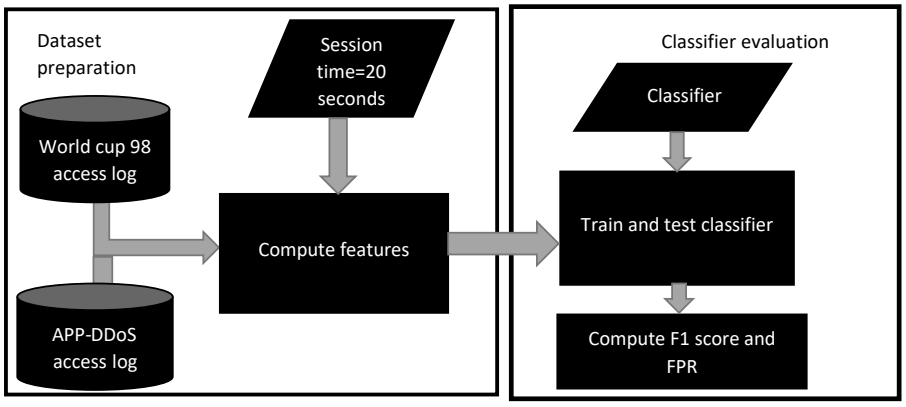


Figure 5.1: Evaluation methodology

5.3 Session time

The third research question investigates *the effect of session time on the performance of the proposed detection system.* All the predictors used by the detection system are computed by considering some time window called session time. The third research question answers the effect of varying session time on the performance of the detection system.

In this research question, we aim to show the trade-off between detection system's response time and performance. If the session time is very small, the response time of the detection system during attack will be lower. However, we also need to check if there is detection performance penalty because of choosing small session time. Studying the effect of session time will help us choose the optimum session time for practical deployment of the detection system.

Evaluation criteria: $F1$ score and FPR are considered to evaluate the performance of classifier on datasets generated by using different session time.

5.3.1 Investigation approach

The evaluation methodology is very similar to the methodology shown in Figure 5.1. However, in this case we generate separate datasets for session time of 20, 40, 60, 120, 180, 240, 300, 360, 420 and 480 seconds. Also, we choose the best classifier obtained from research question two as a classifier and evaluate it on the generated datasets.

5.4 Dataset preparation and Classifier evaluation

In this section, we will discuss details of dataset preparation and classifier evaluation procedures. We will discuss how we prepared datasets from world cup 98 access log and experimentally generated DDoS attack access log. We also discuss the experimental procedures taken to answer research question two and three.

5.4.1 Access log preparation

Data set generation involves data preparation of flash crowds and application layer DDoS attack data. The world cup 98 dataset is used to represent legitimate flash crowds. The DDoS attack is generated by using the same website as a target hosted on a local server.

World cup 98 dataset

World cup 98 dataset (Arlitt and Jin, 1998) consists of all the requests made to the 1998 World Cup Web site (**www.france98.com**) between April 30, 1998 and July 26, 1998. World cup website provided information on the ongoing France 1998 world cup. The website was hosted on multiple servers at different locations. The website received large number of requests from clients who were interested in the world cup game. 1,352,804,107 requests were received by the website during the period. This dataset is used as a flash crowd dataset in this and related researches (Xie and Yu, 2009; Yu et al., 2009, 2012; Bhandari et al., 2016).

The server logs are provided in a binary format. The tools required to process the dataset are also provided (Arlitt and Jin, 1998). World cup 98 dataset is divided in to multiple files with more than one file per day. The number of files depend on the number of requests on that particular day. We have chosen day 66 (June 30, 1998) of the dataset to model flash crowds because it registered maximum number of requests. From the day 66 data, **we have chosen logs of the last three hours of the day because of the game between Argentina and England which took place during those hours**.

Each entry in the server log files represent a single request. The recorded information for each request are timestamp, clientID, objectID,Size, method, status, type and server. Table 5.2 provides description of world cup 98 dataset fields. An example of the log entry is shown below. The request information contains clientID, time stamp, request type and URL of the requested object, HTTP version, response code and replay size respectively from left to right.

104858 - - [30/Jun/1998:21:41:24 +0000] "GET /english/images/-nav_home_off.gif HTTP/1.0" 200 828

Table 5.2: World cup 98 dataset field description

Field	Description
Timestamp	The time stamp refers the time when a request is made. The timestamp has been converted to GMT. The local time in France was 2 hours ahead of GMT
ClientID	A unique integer identifier for the client that issued the request. Each clientID maps to exactly one IP address, and the mappings are preserved across the entire data set. The IP address may also be proxy address. The mapping of id to IP address is not available for privacy reasons.
ObjectID	A unique integer identifier for the requested URL; the mappings of ObjectID with URL are 1-to-1 and are preserved across the entire dataset. The mapping of objectID to actual URL are provided
Size	Reply data size in bytes or zero when the requested file is not found
Method	The client request type GET or POST
Status	HTTP version and response code such as 200 and 404 and so on
Type	Type of requested file based on extension
Server	Identifies which server handled the request

Application layer DDoS attack dataset

To the best of our knowledge, there is no dataset available for application layer DDoS attacks. Hence, the attack data is generated using a DDoS attack tool, BoNeSi (Markus-Go, 2016). The attack was performed on world cup 98 website (**www.france98.com**) hosted locally on closed environment. BoNeSi can generate ICMP, UDP and HTTP flooding attacks from pre-defined botnet size. This tool also accepts URL lists in a file and requests pages randomly. It also generates attack statistics. BoNeSi must be run in a closed environment for correct functionality. In order to achieve that the machine which hosted the BoNeSi tool must get feedback from the target server.

BoNeSi tool is installed in the attack machine and the cached version of world cup website (**www.france98.com**) is hosted on the server machine. Apache web server application is used to host the website. The Attack machine is directly connected to the server machine using cat-6 cable on its network card.

Request flooding and asymmetric attacks are included in the DDoS attack dataset. Request flooding attacks are characterized by high number of requests per machine. Whereas, asymmetric attacks are performed by choosing most resource intensive request. The request rate is usually very low to avoid detection.

Experiment setup

BoNeSi attack tool is installed on Ubuntu 16.04 Linux operating system. In order to conduct attack using BoNeSi tool, the response of the server must be routed back to the attack machine. To achieve this, the IP address of the default gateway of the server must be the same as the IP address of the attack machine. Both server and attack machine are configured as shown in Table 5.3 and 5.4 respectively.

Table 5.3: Server configuration for generating DDoS attack

Component	**Value**
Processor	Intel (R) core i7,3.4GHz
Memory	4GB
Operating system	Windows 10, 64 bits
Web server	Apache version 2.4.23 web server hosted on XAMPP version 3.22
IP address	10.5.10.22
Subnet	255.255.255.0
Default gateway	10.5.10.23

Table 5.4: Attack machine configuration

component	**Value**
Processor	Intel (R) core i3,2.4GHz
Memory	4GB
Operating system	Ubuntu 16.04 LTS, 64 bit
BoNeSi tool	version 2.0
IP address	10.5.10.23
Subnet	255.255.255.0
Default gateway	10.5.10.1

BoNeSi configuration for request flooding attack

Table 5.5 shows the configuration used to generate request flooding application layer DDoS attack. The most important parameters are number of bots or unique source IP address, request rate and URL of requested object. Request flooding attack is generated by sending large number of requests per source IP. This is achieved by limiting the maximum number of bots involved in the attack. BoNeSi provided 50,000 unique number of IP addresses to be used. In order to cover attack scenarios of very small and very large number of bots, 50 bots were taken for small number of bots and 50,000 were taken for large number of bots. URL of requested object is randomly chosen from all web objects in the world cup 98 website.

Table 5.5: Request flooding DDoS attack parameters

Parameter	**Value**
Bots	50,000 unique bots or source IP address
Request rate	500-1000 requests/second
Target URL	Randomly selected object from world cup 98 website
Attack type	TCP (HTTP) flooding

BoNeSi configuration for asymmetric attack

To simulate asymmetric DDoS attacks, 50,000 bots were deployed. The total request rate is lowered so that the number of requests per bot is small. It is difficult to calculate precise request workload. We assume that the server load is proportional to the reply size. This assumption works for static web pages whose contents are retrieved from hard drive. All pages on the world cup 98 website are static pages. Fifty web objects with highest replay size are chosen as a target URL. BoNeSi randomly selects one URL at a time for the request. Table 5.6 shows the configuration parameters used to generate asymmetric attacks.

Table 5.6: Asymmetric DDoS attack parameters

Parameter	**Value**
Bots	50,000 unique bots or source IP address
Request rate	100 requests/second
Target URL	Randomly selected object from 50 web objects with highest page size
Attack type	TCP (HTTP) Asymmetric attack

The attack generation lasted a day. About 1GB of access log data was obtained after conducting the attack for a day. The data included both request flooding attacks and asymmetric attacks.

216.213.134.229 - - [06/Apr/2017:10:04:58 +0300] "GET /images/mpa4606.jpg HTTP/1.0" 200 4375 "-" "Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1; snprtz—S26320700000083—2600#Service Pack 1#2#5#154321—isdn)"

Figure 5.2: Typical attack log request

Figure 5.2 shows an example attack request log entry. The request information contains, IP address, time stamp, request type and requested URL, response code, reply size and the user agent information of the attack respectively from left two right.

5.4.2 Feature computation from server access logs

All candidate classifiers require a numerical input data. Hence, feature computation is required to convert access log data to the numerical dataset. As discussed in Chapter 4, the detection system uses five features or predictors for classification. The features are: *request rate, page popularity, download rate, request inter-arrival time and ratio of successful requests*. All features can be computed using equations discussed in section 4.1 from server access log.

Each web-page of the world cup 98 website has associated page popularity value computed using flash crowd access logs. The page popularity is obtained by summing the number of requests on Day 66 access logs for each page and dividing it by total number of requests. Equation 4.2 is used to compute the page popularity. Page popularity value of each web object is between 0 and 1.

During feature computation, we take the requested web object popularity value. If a client requests more than one web object or more than one request for similar web object in a session time, we use Equation 4.3 to compute the total page popularity.

Session time

For research questions one and two we used 20 second session time for feature computation. For research question 3, we used 20, 40, 60, 120, 180, 240, 300, 360, 420, 480 seconds session time to generate ten datasets.

A feature extraction code is implemented using C++ language. The code is attached in Appendix A. The input to this code is server access logs of both attack and flash crowd. The output of the program is a CSV file. One line in the file represents the information of one client. It contains values of the five features and the label designating if the client is attacker (label value 1) or legitimate flash crowd (label value 0).

For example, the entry [0.35,0.231039,1950.65,15,1,0] in the output file is read as request rate, page popularity, download rate, request inter-arrival time, ratio of successful requests to total number of requests and label respectively.

Combined dataset

We have used *the last three hours of world cup 98 day 66 access log as a flash crowd dataset*. For each session time, the final dataset contains 20,000 examples. From the 20,000 examples, 10,000 are legitimate flash crowd examples and the rest are application layer DDoS attack examples. The generated dataset contains balanced number of attack and normal examples. Both normal examples and attack examples are grouped separately

so shuffling the dataset is required before training and testing the classifiers.

5.4.3 Classifier evaluation

We have considered Decision tree, support vector machine, Naive Gaussian radial basis, random forest and AdaBoost ensemble classifiers for evaluation. All classifiers are implemented in scikit-learn machine learning tool (Pedregosa et al., 2011). **Scikit-learn** is an open source machine learning tool that implements most machine learning algorithms. Scikit-learn is built using python programming language. Scikit-learn is chosen because of its simple and easy to use API's to evaluate machine learning algorithms.

Cross validation

In order to evaluate supervised learning classifiers, k-fold cross validation is used in this research. In k-fold cross validation, the dataset is divided in to k equal parts. In the first round, the first portion will be used for testing and the reset $K-1$ will be used for training. In the second round, the next partition will be used for testing while the rest are for training. This iteration continues K times. The final performance is computed by averaging the performance obtained in all K rounds.

Cross validation is better than splitting the data into training and test data using percentage. Splitting the data using percentage split may bias the test set to either of the classes. If the test split is biased, the performance measure does not reflect the actual performance of the classifier. All classifier performance results in this research were obtained by using *10-Fold* cross validation. Both F1 score and FPR are obtained by taking average of 10 different results.

scikit-learn machine learning tool

Most supervised algorithms in scikit-learn accepts a two-dimensional matrix for training and or testing and a one-dimensional array of the stores the corresponding target value. The row in the two-dimensional training data corresponds to one training instance. The columns represent the feature values of one instance.

All datasets were arranged in such a way that the five features hold the first five columns while the last column is a label indicating whether the instance is DDoS attack or flash crowd.

Support vector machines require scaling so that the values of all features are in the same range. Scaling is performed on the dataset before feeding the data to SVM classifiers. Scikit-learn provides a library for machine learning preprocessing stages including scaling.

The sci-kit function used for the ten-fold cross validation is the **cross_val_score** . This function computes the score of a classifier. The syntax of the function call is

scores = cross_val_score (clf, x, y, cv=10,scoring='f1')

where clf is the candidate classifier, x is a two-dimensional array of examples, y is one dimensional array of target values, CV is the number of chunks the dataset is split in to, CV=10 is used in this experiment, and *scoring* is the scoring parameter that determines the scoring type such as F1, precision and recall.

The return value of the cross_val_score function is an array that contains the score values of each iteration. The number of iteration is determined by the CV value which is 10 in this experiment.

Classifier performance evaluation procedure

For research question two, we used the 20 second dataset and evaluated all candidate classifiers. For research question three, we have used classifier selected in research question two and evaluated the selected classifier with all datasets. The evaluation procedure for both research question is similar and discussed below. The python script used for research question two and research question three is available on Appendix B and Appendix D respectively.

1. Load the dataset from the file in to a two-dimensional array
2. Random shuffle the rows of the two-dimensional array so that positive and negative

samples are mixed randomly

3. Split the dataset in to feature matrix and label
4. Select and initialize the candidate classifier
5. If the classifier is Support vector classifier, the dataset is normalized using scikit-learn scaling function.
6. Compute the F1 score by calling the cross_val_score method
7. Compute the FPR score by calling the cross_val_score

Classification time evaluation procedure

Classification time measures the time it takes for a classifier to classify a given number of examples, 10,000 examples in our case. Classification time measures the computational complexity of classifiers during classification stage. We have compared all candidate classifiers on 20 second dataset. We used 10,000 records for training and 10,000 records for classification to evaluate all classifiers. The experiment is run on the machine whose specification was given in Table 5.3. The evaluation procedure is given below. The python script of this procedure is given in Appendix C.

1. Load the 20 second dataset from the file in to a two-dimensional array
2. Random shuffle the rows of the two-dimensional array so that positive and negative samples are mixed
3. Split the dataset in half, 10,000 training data and 10,000 testing data.
4. Select and initialize the candidate classifier
5. If the classifier is SVM, normalize the training data
6. Train the classifier using the training data
7. Record start time

8. If the classifier is SVM, normalize the test data
9. Classify the test data using the candidate classifier
10. Record end time
11. Compute the time classification time as end time minus start time.
12. Repeat procedures 4 to 11 for every candidate classifier

Feature importance evaluation procedure

In order to practically test feature contributions on research question one, we used random forest classifier to obtain feature importance value for each feature. Feature importance evaluation procedure is discussed below. The python script for this procedure is given in Appendix E.

1. Load the 20 second dataset from the file in to a two-dimensional array
2. Random shuffle the rows of the two-dimensional array so that positive and negative samples are mixed randomly
3. Split the dataset in to feature matrix and label
4. Initialize random forest classifier
5. Train random forest classifier
6. Access the feature importance value

We tried to evaluate our proposed approach in terms of feature contribution and classifier evaluation. We have discussed the details of dataset preparation and experimental setup. We also added experiments to study the effect of session time on the performance of our classification system. We specified F1 score, FPR and classification time as a performance measurement criteria.

Chapter 6

Result and discussion

In this chapter, we present APP-DDoS attack detection system evaluation result and discussion feature importance, APP-DDoS detection and effect of session time on APP-DDoS detection were investigated. The result is discussed in the following subsections.

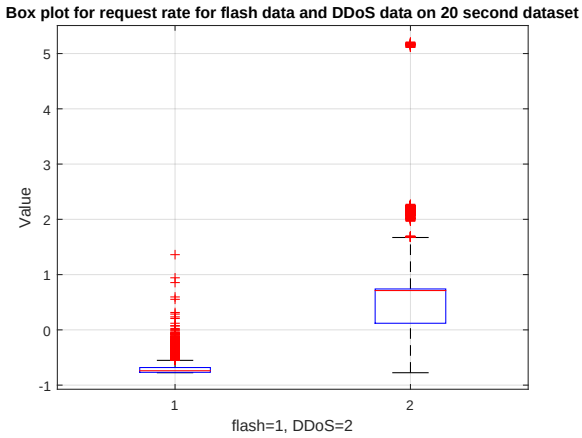
6.1 Feature importance

Figure 6.1 shows the result obtained for the first research question discussed in Section 5.1. The median difference between APP-DDoS and flash is approximately 1 unit for request rate. In addition, there is also an observable difference in outliers. This shows that request rate contribution is potentially higher. The reason for higher feature importance is that Most of the attack data is request flooding attack which is characterized by higher request rate.

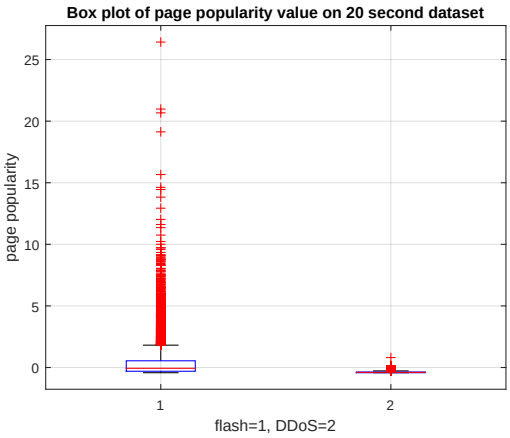
All sub-figures in the Figure 6.1 show box plots of all five features used for detection. All values are normalized to have a mean of 0 and unit variance. The lower line of box plot represents the 25% or first quartile. The top of the box represents the third quartile. The line inside the box indicates the median. Values labeled in '+' sign are outliers.

When we see the page popularity in Figure 6.1b box plot, the median of flash and APP-DDoS is very close and difficult to separate. This means that using page popularity only, it is difficult to separate APP-DDoS attack from flash crowds.

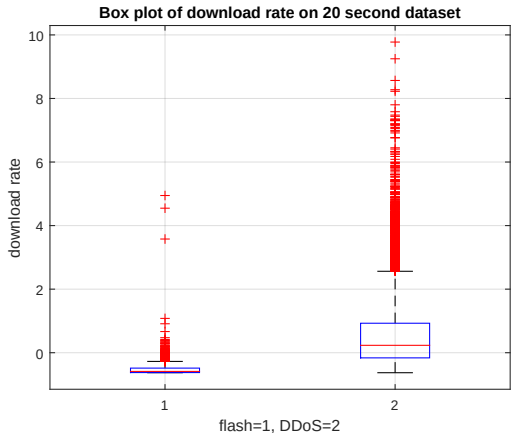
For the case of download rate in Figure 6.1c, the median difference between APP-DDoS and flash is approximately 0.8 units. Due to the asymmetric APP-DDoS attack, there are many outliers observed on the APP-DDoS box plot. The median difference is close to that of request rate.



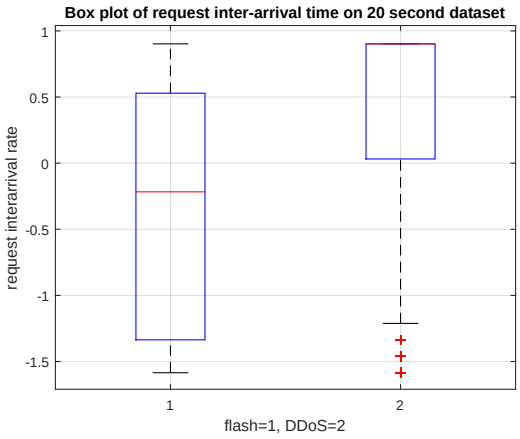
(a) Request rate box plot



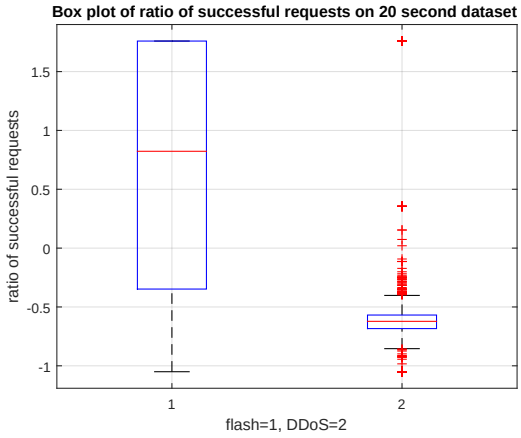
(b) page popularity box plot



(c) Download rate box plot



(d) Request inter-arrival time box plot



(e) Ratio of successful requests box plot

Figure 6.1: Box plots of all features used for detection

Figure 6.1d shows the box plot of request inter-arrival time (RIA). The difference between the median of APP-DDoS and flash was approximately 1.05 unit. however, there is high overlap between flash and APP-DDoS boxes. As a result, the potential contribution of

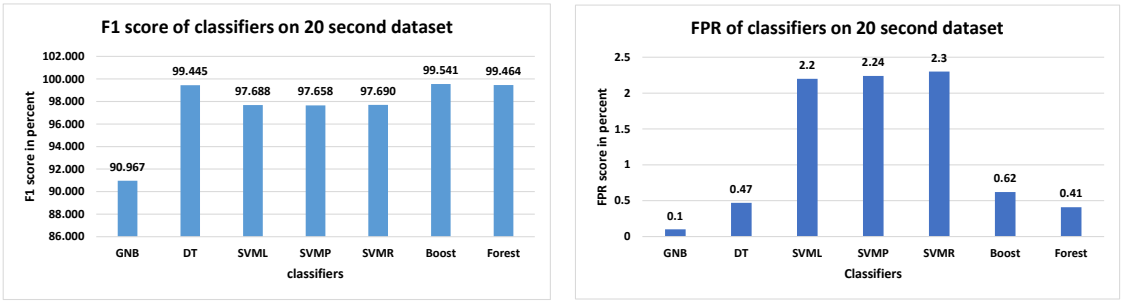
RIA for detection is very low.

We see the highest median difference between flash and APP-DDoS, in Figure 6.1e, for ratio of successful requests (RSR) approximately 1.35 units. The big median difference occurred because legitimate users in flash crowds request pages by following links which increases the probability if the request being successful, 200 response code. But APP-DDoS attacks select pages randomly which reduces the probability of the request being successful.

Although, all five features are used for detection, request rate and download rate have higher contribution for detection among the five features based on qualitative analysis.

6.2 APP-DDoS detection

As a second research question, we investigated whether we can distinguish APP-DDoS attacks from legitimate user flash crowd. We have compared six classifiers that can be used in the detection system. We have measured the classification performance as well as computational performance of the candidate classifiers.



(a) F1 score of candidate classifiers

(b) FPR of candidate classifiers

Figure 6.2: The classification performance of all candidate classifiers in terms of F1 score and FPR. Both F1 and FPR are given in percentage. The classifier names are read as GNB(Gaussian Naive Bayes), DT (Decision tree), SVML (SVM with linear kernel), SVMP (SVM with polynomial kernel), SVMR (SVM with radial basis kernel), Boost (AdaBoost) and Forest (Random forest)

Figure 6.2 shows the F1 score and FPR score of candidate classifiers on 20 second dataset. GNB classifier showed the lowest F1 score of 90.97%. When there exists correlation

between features, the assumption of GNB that there is no correlation among features fails which explains the result. however, the lowest FPR was obtained by GNB. When lower F1 score is accompanied by lower FPR, it implies that most of the time the classifier guesses the input as flash crowd or negative.

As we can see Figure 6.2a, DT, Boost and Forest had F1 score higher than 99%. Boost had the highest F1 score of 99.541%. But the F1 score of DT and Forest are also very close. We performed a statistical test whether the difference among scores of Forest, Boost and DT are statistically significant.

We took the 10 F1 score values, obtained during 10-fold cross validation, of DT, Boost and Forest and performed analysis of variance (ANOVA) test. ANOVA test showed that the difference among the three classifiers is not statistically significant with 95% confidence. The ANOVA calculation is available on Appendix F. The same is true when we do ANOVA on FPR score of the three classifiers. The ANOVA calculation is available on Appendix G. This implies that *we can choose any classifier for our APP-DDoS detection among DT, Boost and Forest.*

To see the effect of kernels on SVM performance, we have used ANOVA to test the difference among F1 score of linear, polynomial and radial basis kernels. ANOVA test showed that the difference among the three kernels is not statistically significant with 95% confidence. The ANOVA calculation is available on Appendix H.

Figure 6.3 shows the classification time of candidate classifiers. DT was the fastest classifier with 0.37 millisecond. The reason for fast classification time of DT is that classification in DT is just traversing a tree. DT training has techniques to make the decision tree depth as small as possible. Traversing small depth trees requires small time. GNB is the second fastest with 0.721 milliseconds. The classification algorithm is relatively simpler compared to other classifiers. SVM and Boost took high classification time. SVM's require scaling of feature values which makes classification time longer compared to DT and GNB. Even though, Boost does not require scaling, Boost has to

make fifty iterations to classify one example. This makes the classification time higher.

The kernel choice had big effect on the classification time of SVM. The computational complexity of SVM during classification is dependent on the complexity of the kernel. Radial basis kernel took 891.407 milliseconds while linear kernel took 69.68 milliseconds.

We have seen that DT, Forest and Boost showed comparable F1 score and FPR. But the classification time of DT is much smaller than Forest and Boost. APP-DDoS detection system must be computationally efficient not to contribute to the already exhausted server resources by APP-DDoS and/or flash crowd. DT has bigger advantage compared to Boost and Forest when we consider classification time. That makes **decision tree recommended classifier to distinguish between flash crowd and APP-DDoS attack**.

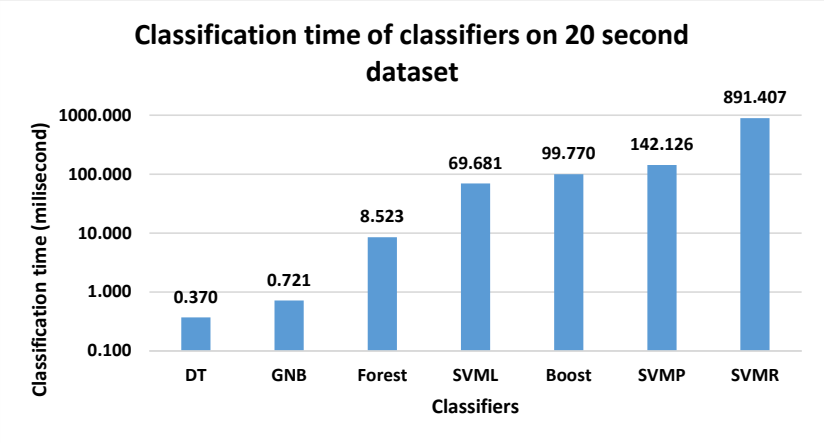


Figure 6.3: Classification time of candidate classifiers. Classification time measured the time it took classifiers to classify 10,000 examples in milliseconds. The training time was not included in this measurement.

Feature contribution using random forest

We have provided qualitative analysis on feature importance in the result of research question one. To practically test feature contribution on our dataset, we used random forest classifier feature importance value. Random forest shows the portion of dataset classified by using a particular feature during training.

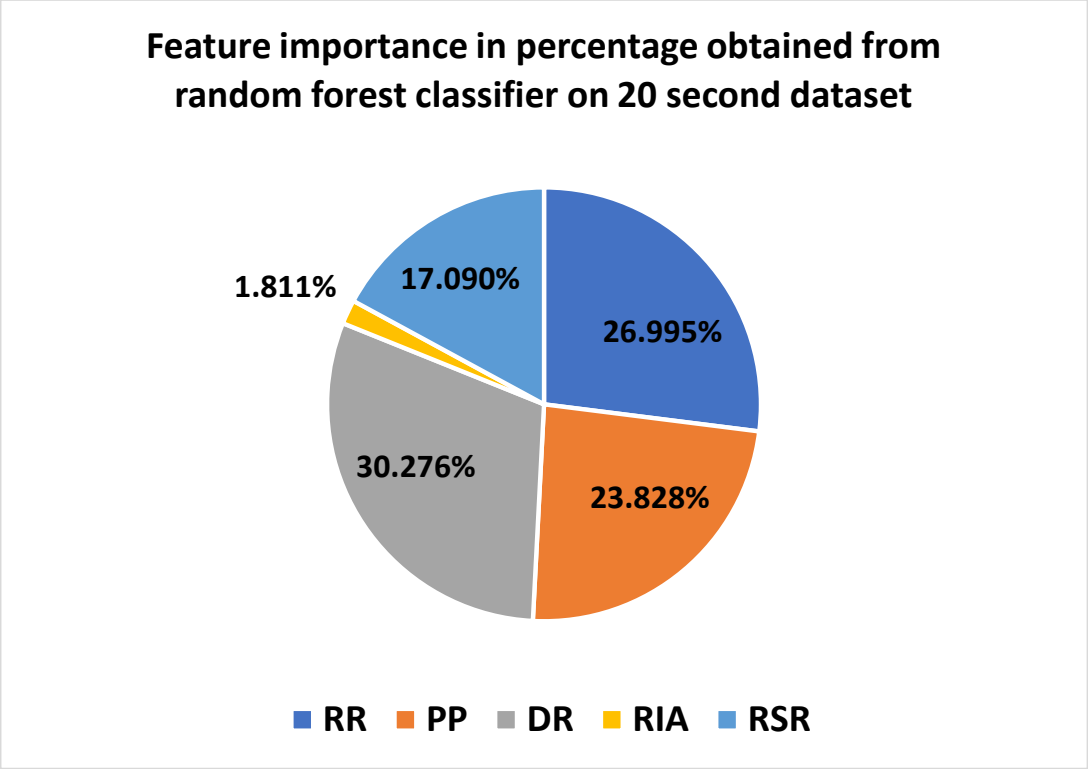


Figure 6.4: Random forest feature importance result

Figure 6.4 shows the feature importance of all five features obtained from random forest classifier. The result obtained shows that download rate has the highest contribution of 30.276% and request rate is second with contribution of 26.995%. This result supports the qualitative analysis of feature importance discussed in Section 6.1.

It is possible to distinguish between APP-DDoS and flash crowd using our proposed approach. Our proposed approach uses supervised classifiers for detection. By comparing six candidate classifiers, we have shown that ***decision tree classifier*** *is best suited to be used as APP-DDoS detection algorithm in our proposed approach.*

6.3 Effect of session time

As discussed in Section 6.2, decision tree classifier is selected for APP-DDoS detection. Research question three deals with the effect of session time on classification performance of classifiers. To investigate effect of session time, we tested decision tree classifier on datasets generated using 20, 40, 60, 120, 180, 240, 300, 360, 420 and 480 seconds session time.

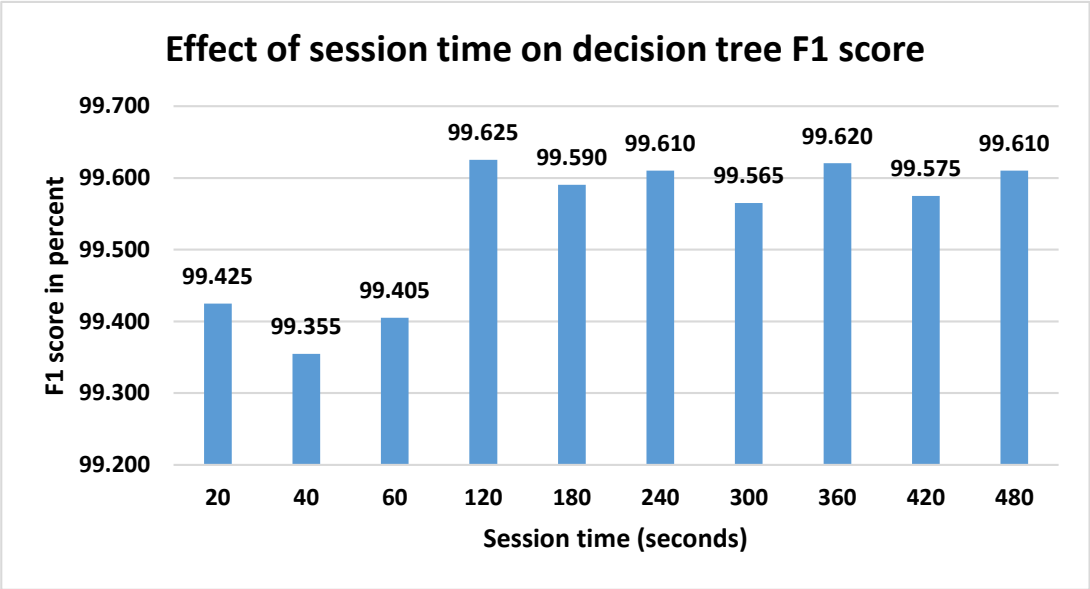


Figure 6.5: Effect of session time on F1 score of decision tree classifier

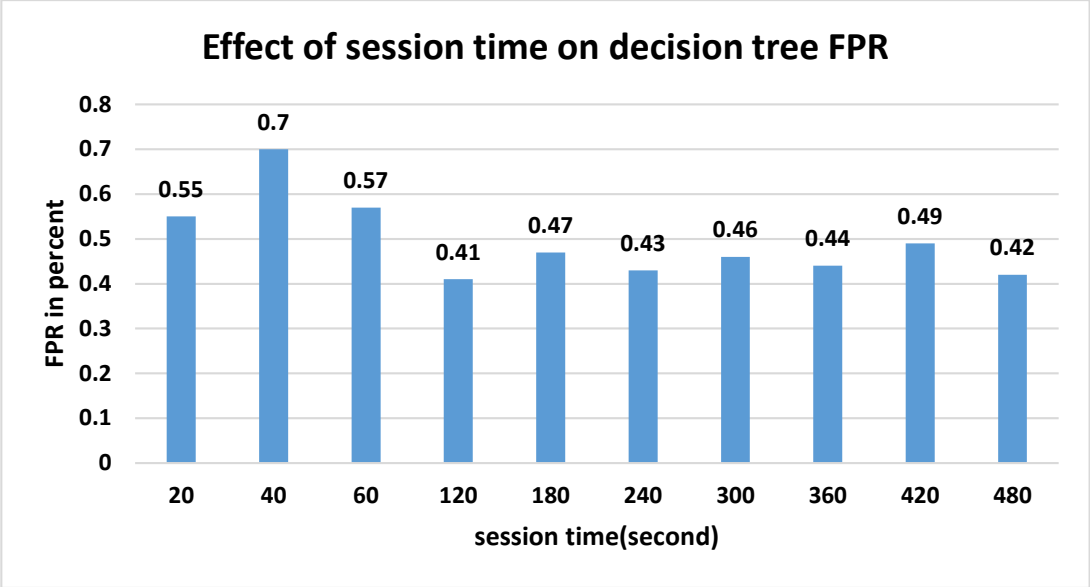


Figure 6.6: Effect of session time on the performance of decision tree classifier

Figure 6.5 shows the effect of session time on F1 score of decision tree classifier. The

highest F1 score was observed for 120 second session time. The difference between the highest and lowest F1 score is 0.275 %. This shows that the effect of session time on the F1 score is very small. But with some fluctuation, F1 score has a tendency to increase as session time increases.

Figure 6.6 shows the effect of session time on FPR score of decision tree classifier. The FPR has even smaller variation among all session times. The difference of FPR among all session time is not statistically significant when we applied ANOVA. The ANOVA calculation is available on Appendix I.

Session time has direct implication on the response time of the detection system. If the session time is smaller, then the detection system can respond quickly. It is better to choose smaller session time by losing some accuracy than choosing higher session time when the impact of the attack is high. When we see the difference between F1 score of 120 second, highest F1 score, and 20 second session time, it was only 0.2% and the FPR difference is 0.14%. As we can see, there is very little advantage gained by using 120 second session time compared to 20 second.

For the smallest session time, 20 second, decision tree has F1 score of 99.425% and FPR of 0.55%. Based on the result obtained, we recommend 20 second a session time for feature computation in our detection system.

6.4 Threats to validity

Internal threats to validity

Threats to internal validity are mainly caused by variation in instrumentation, and effect due to uncontrolled variables. We run all experiments on the same computer to avoid instrumental variation. The effect of uncontrolled variables is mainly observed when we measured the classification time of classifiers. The classification time may be affected by external concurrent processes that run on the same computer at the time of the experiment. To address this threat, we have repeated the measurement ten times and

took the average. In addition, we closed non-vital applications during experiment.

External threats to validity

We have evaluated our proposed approach on specific flash crowd dataset and using only one attack tool. In addition, the flash crowd dataset is old which may not represent current flash crowds. Those are major threat to external validity. But our proposed approach is independent of the data. We can test our approach on any dataset. We will reevaluate our approach when current flash crowd dataset is available. In addition, we can test our approach on any website without changing our detection system.

The other problem is that we only found one DDoS attack tool suitable for our research. But the tool is very flexible with many configurable parameters. We tried to approximate the functionality of other DDoS attack tools by manipulating the configuration. This makes the attack tool more representative.

Construct threats to validity

The main threats to construct validity occur during choice of features for detection and during choice of classifiers. For example, we have evaluated six classifiers, But the best classifier may not be among the candidate classifiers. To minimize threats of construct validity because of classifier selection, we selected representative examples from most commonly used supervised machine learning algorithms. Most other supervised classifiers are derivatives of the candidate classifiers.

The other potential threat to construct validity is the choice of features. We did not consider all possible features for detection. The reason for this is that if we choose a feature that cannot be computed from our dataset, it is difficult to evaluate our proposed approach. But we have obtained very good result using only six features by logically choosing features that have higher contribution for detection among features suggested in related litratures.

Conclusion threat to validity

The main threats to conclusion validity are too small sample size, measurement error and violation of assumption in test statistics. We have 20,000 examples for both flash and APP-DDoS in our dataset. When we observe both APP-DDoS attack and flash crowd dataset, the feature values are similar or very close to each other. We believe that our dataset sample size is not small for our problem. Since the measurement and experiment was done on computers, the measurement error only comes from computation errors from machines. But we do not need high precise measurement. Hence, the measurement error is negligible. We have used analysis of variance (ANOVA) as test statistics. The assumption of ANOVA is that the data must be normally distributed. In order not to violate this assumption, we tested our data for normality using Kolmogorov-Smirnov test.

Chapter 7

Conclusion and recommendation

7.1 Conclusion

This research tried to address the problem of identifying application layer DDoS attacks from legitimate flash crowds. We proposed a supervised machine learning based detection system that uses request rate, page popularity, download rate, request inter-arrival time and ratio of successful requests as features that distinguish between APP-DDoS attack and flash crowds. We evaluated our proposed approach using flash crowd and APP-DDoS attack dataset. The evaluation of the proposed approach raises three research questions. The first question is the investigation of contribution of each feature for detection. The second research question is that can we identify APP-DDoS attacks from flash crowds using our proposed approach. All the features are calculated by considering a session time. So, what should be the value of session time for effective detection is the third research question. We have evaluated six supervised classifiers on our dataset. We have selected F1 Score and false positive rate as a classification performance evaluation criteria and classification time as computational complexity evaluation criteria to compare the classifiers.

Download rate had the highest contribution for detection with request rate and page popularity following. It is possible to identify APP-DDoS attack from flash crowd with our proposed approach. Decision tree performed best among candidate classifiers considering F1 score, FPR and classification time as evaluation criteria. Decision tree classifier had 99.445% F1 score, 0.47% FPR and the smallest classification time of 0.37 milliseconds. This shows that decision tree is a good candidate for our detection system. Variation of session time has very small impact on the performance of decision tree classifier. The smallest session time we have tested is 20 second. The difference between the 20 second

F1 score and 120 second, session time with best F1 score, is very small. In addition, the difference between FPR scores of 20 and 120 seconds session time is not statistically significant. This implies that we can choose any session time with very small impact on performance of our detection system.

7.2 Recommendation

Based on the result obtained, the problem of distinguishing between APP-DDoS and flash crowds can be addressed by using decision tree based detection system. Decision tree has high classification accuracy and low computational complexity which is desirable in network security problems. Regarding the session time, it is recommended to choose session time based on the severity of APP-DDoS attack. If the attack is severe, smaller session time is recommended for faster response and attack mitigation.

Future work

The main limitation of the research is the unavailability of latest dataset of flash crowds. We were forced to use the world cup 98 dataset. Which is the standard application layer flash crowd dataset up to now even though it was recorded before 19 years. Our proposed approach should be tested on latest dataset for more concrete and applicable result. The second limitation was the unavailability of APP-DDoS dataset which forced us to use DDoS attack generation tool. Based on the aforementioned limitation of this research, we recommend the following points to be addressed as a future work.

- The proposed approach should be tested on a new dataset that contains examples of real flash crowds and DDoS attacks.
- There is no standard criteria to generate a application layer DDoS attack in simulation. Some standard should be set on how to generate AppDDoS attack that closely resembles real attacks. This can be done by analyzing patterns of real world APP-DDoS attacks.

- The proposed detection system should also be tested in real time by setting up a simulation in a controlled environment.

Bibliography

Akamai (2016). State of the internet/security report q4 2016. `https://content.akamai.com/pg7967-q4-soti-security-report.html`.

Alomari, E., Manickam, S., Gupta, B., Karuppayah, S., and Alfaris, R. (2012). Botnet-based distributed denial of service (ddos) attacks on web servers: classification and art. *arXiv preprint arXiv:1208.0403*.

Arlitt, M. and Jin, T. (1998). 1998 world cup web site access logs.

Beitollahi, H. and Deconinck, G. (2014). Connectionscore: a statistical technique to resist application-layer ddos attacks. *Journal of Ambient Intelligence and Humanized Computing*, 5(3):425–442.

Bhandari, A., Sangal, A. L., and Kumar, K. (2016). Characterizing flash events and distributed denial-of-service attacks: an empirical investigation. *Security and Communication Networks*.

Fielding, R., Gettys, J., Mogul, J., Frystyk, H., Masinter, L., Leach, P., and Berners-Lee, T. (1999). Hypertext transfer protocol–http/1.1. Technical report.

Fitsum, A. (2008). Distributed denial of service attack detection: A hybrid intelligent system approach.

Freund, Y. (1990). Boosting a weak learning algorithm by majority. In *COLT*, volume 90, pages 202–216.

Freund, Y., Schapire, R. E., et al. (1996). Experiments with a new boosting algorithm. In *icml*, volume 96, pages 148–156.

Garg, A. and Reddy, A. N. (2001). Mitigating denial of service attacks using qos regulation. In *In Proceedings of International Workshop on Quality of Service (IWQoS*. Citeseer.

Graf, A. B. and Borer, S. (2001). Normalization in support vector machines. In *Pattern Recognition: 23rd DAGM Symposium, Munich, Germany, September 12-14, 2001. Proceedings*, page 277. Springer.

Kandula, S., Katabi, D., Jacob, M., and Berger, A. (2005). Botz-4-sale: Surviving organized ddos attacks that mimic flash crowds. In *Proceedings of the 2nd conference on Symposium on Networked Systems Design & Implementation-Volume 2*, pages 287–300. USENIX Association.

Kumar, R., Arun, P., and Selvakumar, S. (2009). Distributed denial-of-service (ddos) threat in collaborative environment-a survey on ddos attack tools and traceback mechanisms. In *Advance Computing Conference, 2009. IACC 2009. IEEE International*, pages 1275–1280. IEEE.

Li, K., Zhou, W., Li, P., Hai, J., and Liu, J. (2009). Distinguishing ddos attacks from flash crowds using probability metrics. In *Network and System Security, 2009. NSS'09. Third International Conference on*, pages 9–17. IEEE.

Maier, G., Feldmann, A., Paxson, V., and Allman, M. (2009). On dominant characteristics of residential broadband internet traffic. In *Proceedings of the 9th ACM SIGCOMM conference on Internet measurement conference*, pages 90–102. ACM.

Markus-Go (2016). Bonesi - the ddos botnet simulator. `https://github.com/Markus-Go/bonesi`.

Metsis, V., Androutsopoulos, I., and Paliouras, G. (2006). Spam filtering with naive bayes-which naive bayes? In *CEAS*, volume 17, pages 28–69.

Norman, J. (2016). The first documented denial of service attack.

Patil, M. M. and Kulkarni, U. (2011). Mitigating app-ddos attacks on web servers. *International Journal of Computer Science and Information Security*, 9(7):40.

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau,

D., Brucher, M., Perrot, M., and Duchesnay, E. (2011). Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830.

Ramamoorthi, A., Subbulakshmi, T., and Shalinie, S. M. (2011). Real time detection and classification of ddos attacks using enhanced svm with string kernels. In *Recent Trends in Information Technology (ICRTIT), 2011 International Conference on*, pages 91–96. IEEE.

Ranjan, S., Swaminathan, R., Uysal, M., and Knightly, E. W. (2006). Ddos-resilient scheduling to counter application layer attacks under imperfect detection. In *INFOCOM*. Citeseer.

Sachdeva, M., Kumar, K., and Singh, G. (2016). A comprehensive approach to discriminate ddos attacks from flash events. *Journal of Information Security and Applications*, 26:8–22.

Specht, S. M. and Lee, R. B. (2004). Distributed denial of service: Taxonomies of attacks, tools, and countermeasures. In *ISCA PDCS*, pages 543–550.

Ting, S., Ip, W., and Tsang, A. H. (2011). Is naive bayes a good classifier for document classification. *International Journal of Software Engineering and Its Applications*, 5(3):37–46.

Wang, J., Yang, X., and Long, K. (2010). A new relative entropy based app-ddos detection method. In *Computers and Communications (ISCC), 2010 IEEE Symposium on*, pages 966–968. IEEE.

Xie, Y. and Yu, S.-Z. (2006). A novel model for detecting application layer ddos attacks. In *Computer and Computational Sciences, 2006. IMSCCS'06. First International Multi-Symposiums on*, volume 2, pages 56–63. IEEE.

Xie, Y. and Yu, S.-Z. (2009). Monitoring the application-layer ddos attacks for popular websites. *IEEE/ACM Transactions on networking*, 17(1):15–25.

Xu, C., Zhao, G., Xie, G., and Yu, S. (2014). Detection on application layer ddos using random walk model. In *2014 IEEE International Conference on Communications (ICC)*, pages 707–712. IEEE.

Yadav, S. and Selvakumar, S. (2015). Detection of application layer ddos attack by modeling user behavior using logistic regression. In *Reliability, Infocom Technologies and Optimization (ICRITO)(Trends and Future Directions), 2015 4th International Conference on*, pages 1–6. IEEE.

Ye, C. and Zheng, K. (2011). Detection of application layer distributed denial of service. In *Computer Science and Network Technology (ICCSNT), 2011 International Conference on*, volume 1, pages 310–314. IEEE.

Yu, J., Li, Z., Chen, H., and Chen, X. (2007). A detection and offense mechanism to defend against application layer ddos attacks. In *Networking and Services, 2007. ICNS. Third International Conference on*, pages 54–54. IEEE.

Yu, S., Thapngam, T., Liu, J., Wei, S., and Zhou, W. (2009). Discriminating ddos flows from flash crowds using information distance. In *NSS 2009: Proceedings of the third International Conference on Network and System Security*, pages 351–356. IEEE.

Yu, S., Zhou, W., Jia, W., Guo, S., Xiang, Y., and Tang, F. (2012). Discriminating ddos attacks from flash crowds using flow correlation coefficient. *IEEE Transactions on Parallel and Distributed Systems*, 23(6):1073–1080.

Appendix A

Feature computation C++ source code

```
#include <iostream>


#include <vector>
#include <fstream>
#include <cmath>
#include <string>
#include <stdio.h>
#include <stdlib.h>
#include <sstream>
#include <locale>
#include <boost/algorithm/string.hpp>
#include <boost/algorithm/string/predicate.hpp>
#include <boost/lexical_cast.hpp>
#include <boost/date_time/posix_time/posix_time.hpp>
#include "AssocativeArray.hpp"
/*namespace*/
using namespace boost;
using namespace posix_time;
using namespace std;
/*function declaration*/
void load_Popularity(const string popularityFile);
ptime parseFromString(string _time);
void read_Log(string logFile, int session_time, string outputFileName);
/*Structure dec*/
struct Feature {
```

```
        double num_request; // number of requests per session
        double num_success_requests; //number of successful requests
        double page_popularity; // sum of page popularities
        double download_rate; // sum of replay size
        time_duration client_session_time; // time between first and last
            ↪ request
        ptime current_time; //holds current time for request interarrival
            ↪ calculation
        ptime firstTime; // stores the first request time
        time_duration request_inter_arrival; // request inter arrival
};
struct WebObject {
        int id; //object Id
        double popularity;
};

/*Web object address*/
AssocArray<WebObject> object; // stores popularity of each url or object
/*Vector of feature structure*/
AssocArray<Feature> feature; //stores extracted features from the logs

int main() {
        string log;
        int session_time;
        cout << "Enter␣the␣log␣file␣name" << endl;
        cin >> log;
        cout << "Enter␣the␣session␣time␣(seconds)" << endl;
        cin >> session_time;
        load_Popularity("finalPopFile.csv");
```

```
        read_Log(log, session_time, "dataset.csv");


        return 0;
}
ptime parseFromString(string _t) {
        string _time = _t;
        trim_left_if(_time, is_any_of("["));
        ptime t;
        try {
                stringstream ss(_time.c_str());
                time_input_facet* timefacet = new time_input_facet("%d/%b
                    ↪ /%Y:%H:%M:%S");
                ss.imbue(locale(locale::classic(), timefacet));
                // turn on error
                ss.exceptions(std::ios_base::failbit);
                ss >> t;
        } catch (std::exception& e) {
                cout << e.what() << endl;
        }
        return t;


}
void load_Popularity(const string popularityFile) {
        string url[90000]; // store URL
        fstream file;
        file.open("object_mappings.sort");


        if (!file.is_open()) {
                cout << "File␣not␣found␣in␣the␣directory" << endl;
```

```
        exit(EXIT_FAILURE);
    }

    string _id, urlName, line;
    int id;

    while (getline(file, line)) {
        trim(line);
        vector<string> values;
        split(values, line, is_any_of("␣"));

        try {
            id = boost::lexical_cast<int>(values[0]);

        } catch (boost::bad_lexical_cast & e) {
            std::cout << "Exception␣caught␣:␣" << e.what() <<
                ↪ _id << std::endl;
        }
        url[id] = values[1];
    }

    file.close();

    file.open(popularityFile.c_str());
    if (!file.is_open()) {
        cout << popularityFile << "␣not␣found␣in␣the␣directory" <<
            ↪ endl;
        exit(EXIT_FAILURE);
    }
```

```
//read popularity from the file

while (getline(file, line)) {
        //split the line by ","
        vector<string> values;
        trim(line);
        split(values, line, is_any_of(","));
        //build a structure
        WebObject wo;
        //cout << values[0]<<","<<values[1] <<endl;
        try {
                wo.id = boost::lexical_cast<int>(values[0]);
                //cout << url[wo.id] << endl;
        } catch (boost::bad_lexical_cast & e) {
                std::cout << "Exception␣caught␣:␣" << e.what() <<
                    ↪ _id << std::endl;
        }

        try {
                wo.popularity = boost::lexical_cast<double>(values
                    ↪ [1]);

        } catch (boost::bad_lexical_cast & e) {
                std::cout << "Exception␣caught␣:␣" << e.what() <<
                    ↪ _id << std::endl;
        }
        object.AddItem(url[wo.id], wo);
```

```
        }
        file.close();
}

void read_Log(string logFile, int session_time, string outputFileName) {
        //create output file
        ofstream output(outputFileName.c_str(), fstream::out | fstream::
            ↪ app);
        if (output.is_open()) {
                cout << "output␣file␣ready␣to␣run" << endl;
        } else {
                cout << "output␣file␣failed" << endl;
        }

        //1.Open server log file
        fstream file;
        file.open(logFile.c_str());

        if (!file.is_open()) {
                cout << logFile << "␣not␣found␣in␣the␣directory" << endl;
                exit(EXIT_FAILURE);
        }
        int count = 1;
        // read through the log file
        string line;

        //file>>_id>>_user>>_password>>_time>>_offset>>_request>>_url>>
            ↪ _http>>_code>>_reply
        ptime sessionEnd; // used to track session end time
```

```
// read first request for initialization
getline(file, line);
vector<string> token;
trim(line);
split(token, line, is_any_of("␣"));

Feature f;
f.current_time = parseFromString(token[3]);
f.firstTime = f.current_time; // assign current time as first
    ↪ time
sessionEnd = f.current_time + seconds(session_time);
if (starts_with(token[8], "2")) {
        //success code
        f.num_success_requests = 1;
}
try {
        f.download_rate = lexical_cast<double>(token[9]);
        f.num_request = 1;
        f.request_inter_arrival = seconds(0);
        if (object.IsItem(token[6])) {
                f.page_popularity = object[token[6]].popularity;
        } else {
                f.page_popularity = 0.0;
        }
        feature.AddItem(token[0], f);

} catch (boost::bad_lexical_cast & e) {
        std::cout << "Exception␣caught␣:␣" << e.what() << endl;
```

```
}

//continue reading file
while (getline(file, line)) {

        vector<string> token;
        trim(line);
        split(token, line, is_any_of("␣"));
        ptime t = parseFromString(token[3]);

        if (t > sessionEnd) {
                sessionEnd = parseFromString(token[3]) + seconds(
                    ↪ session_time);
                //dump the data to csv file
                for (int i = 0; i < feature.Size(); i++) {
                        Feature f = feature[feature.GetItemName(i)];
                        time_duration session = f.current_time - f.
                            ↪ firstTime;
                        output << f.num_request / session_time << ",
                            ↪ "
                                        << f.page_popularity << ","
                                        << f.download_rate /
                                            ↪ session_time << "," <<
                                            ↪ ","
                                        << f.num_success_requests / f.
                                            ↪ num_request << ","
                                        << session.seconds() << ",1"
                                            ↪ << endl;
                }
```

```
                feature.clearData();
        } else {
                //search if the request exists
                if (feature.IsItem(token[0])) {
                        try {

                                if (starts_with(token[8], "2")) {
                                        //success code
                                        feature[token[0]].
                                            ↪ num_success_requests++;
                                        feature[token[0]].
                                            ↪ download_rate +=
                                            ↪ lexical_cast<double>(
                                                        token[9]);
                                }

                                feature[token[0]].num_request++;
                                feature[token[0]].
                                    ↪ request_inter_arrival += t
                                                - feature[token[0]].
                                                    ↪ current_time;
                                feature[token[0]].current_time = t;
                                if (object.IsItem(token[6])) {
                                        feature[token[0]].
                                            ↪ page_popularity +=
                                                        object[token
                                                            ↪ [6]].
                                                            ↪ popularity
                                                            ↪ ;
```

```
                }

            } catch (boost::bad_lexical_cast & e) {
                std::cout << "Exception␣caught␣:␣" <<
                    ↪ e.what() << endl;
            }

        } else {

            try {
                Feature f;
                f.current_time = parseFromString(
                    ↪ token[3]);
                f.firstTime = f.current_time;
                if (starts_with(token[8], "2")) {
                    //success code
                    f.num_success_requests = 1;
                    f.download_rate = lexical_cast
                        ↪ <double>(token[9]);
                } else {
                    f.download_rate = 0;
                    f.num_success_requests = 0;
                }

                f.num_request = 1;
                f.request_inter_arrival = seconds(0);
                if (object.IsItem(token[6])) {
                    f.page_popularity = object[
                        ↪ token[6]].popularity;
```

```
                                } else {
                                        f.page_popularity = 0.0;
                                }

                                feature.AddItem(token[0], f);

                        } catch (boost::bad_lexical_cast & e) {
                                std::cout << "Exception␣caught␣:␣" <<
                                    ↪ e.what() << endl;
                        }

                }
        }

        count++;

        if (count % 100000 == 0) {
                cout << count << "requests␣processed" << endl;
        }

    }
    output.close(); //close the dataset file
}
```

```
/*
 * AssocativeArray.hpp
 *
 * Created on: 30 Jan 2011
 * Author: arnavguddu
```

```
 * https://www.codeproject.com/Articles/149879/Associative-Array-in-C
 */


#ifndef SRC_ASSOCATIVEARRAY_HPP_
#define SRC_ASSOCATIVEARRAY_HPP_









#endif /* SRC_ASSOCATIVEARRAY_HPP_ */
#include <iostream>
 #include <vector>
 using namespace std;


 template <class T>
 class AssocArray
 {
    private:
        typedef struct _Data
        {
            T data;
            string name;
        } Data ;
        vector<Data> stack;
    public:
        long Size()
        {
            return stack.size();
```

```
}
void clearData(){
        stack.clear();
}

int IsItem(string name)
{
    for(int i=0; i<Size(); i++)
    {
        if(stack[i].name == name)
            return true;
    }
    return false;
}

bool AddItem(string name, T data)
{
    if(IsItem(name))
        return false;
    Data d;
    d.name = name;
    d.data = data;
    stack.push_back(d);
    return true;
}

T& operator [] (string name)
{
    for(int i=0; i<Size(); i++)
```

```
    {
        if(stack[i].name == name)
            return stack[i].data;
    }
    long idx = Size();
    Data d;
    d.name = name;
    stack.push_back(d);
    return stack[idx].data;
}

string GetItemName(long index)
{
    if(index<0)
        index = 0;
    for(int i=0; i<Size(); i++)
        if(i == index)
            return stack[i].name;
    return "";
}

T& operator [] (long index)
{
    if(index < 0)
        index = 0;
    for(int i=0; i<Size(); i++)
    {
        if(i == index)
            return stack[i].data;
```

```
            }
            return stack[0].data;
        }

};
```

Appendix B

Python script for classifier evaluation

```
import numpy as np
from sklearn.metrics import make_scorer
from sklearn.model_selection import cross_val_score
from sklearn import preprocessing
from sklearn import tree
from sklearn.ensemble import AdaBoostClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.naive_bayes import GaussianNB
from sklearn import svm
filePath="/home/biruk/workspace/DDoS expt/combined/";
fileNames=["20sec.csv"];#"2min.csv","3min.csv","4min.csv","5min.csv","6
    ↪ min.csv","7min.csv","8min.csv"];
#open the output file
outputFile=open('/home/biruk/workspace/DDoS expt/combined/statistics.csv'
    ↪ ,'w');
def false_positive_rate (truth,pred):
    FP = np.sum(np.logical_and(pred == 1, truth == 0));
    TN = np.sum(np.logical_and(pred == 0, truth == 0));
    return FP*1.0/(FP+TN)

fpr_score=make_scorer(false_positive_rate);#greater is better true
def printArray(outputFile,array):
    for x in np.nditer(array):
        outputFile.write(str(x)+",");
    outputFile.write("\n");
```

```
def evaluate_classifier(fileName,ofObj):
    data= np.genfromtxt(filePath+fileName,delimiter=',');
    np.random.shuffle(data);#reorder the samples
    x=data[:,[0,1,2,3,4]]; #the feature matrix
    y=data[:,5]; #the target value as an array
    xn=preprocessing.scale(x);
    ofObj.write(fileName+"\n");
    ofObj.write("GNB,");
    clf=GaussianNB();
    scores=cross_val_score(clf, x, y, cv=10,scoring='f1')
    printArray(ofObj,scores);
    scores=cross_val_score(clf, x, y, cv=10,scoring=fpr_score)
    printArray(ofObj,scores);

    ofObj.write("DT,");
    clf = tree.DecisionTreeClassifier();
    scores=cross_val_score(clf, x, y, cv=10,scoring='f1')
    printArray(ofObj,scores);
    scores=cross_val_score(clf, x, y, cv=10,scoring=fpr_score)
    printArray(ofObj,scores);

    ofObj.write("SVML,");
    clf = svm.SVC( kernel='linear')
    scores=cross_val_score(clf, xn, y, cv=10,scoring='f1')
    printArray(ofObj,scores);
    scores=cross_val_score(clf, xn, y, cv=10,scoring=fpr_score)
    printArray(ofObj,scores);

    ofObj.write("SVMP,");
```

```
clf = svm.SVC( kernel='poly')
scores=cross_val_score(clf, xn, y, cv=10,scoring='f1')
printArray(ofObj,scores);
scores=cross_val_score(clf, xn, y, cv=10,scoring=fpr_score)
printArray(ofObj,scores);

ofObj.write("SVMR,");
clf = svm.SVC( kernel='rbf');
scores=cross_val_score(clf, xn, y, cv=10,scoring='f1');
printArray(ofObj,scores);
scores=cross_val_score(clf, xn, y, cv=10,scoring=fpr_score);
printArray(ofObj,scores);

ofObj.write("Boost,");
clf =AdaBoostClassifier(n_estimators=50);
scores=cross_val_score(clf, x, y, cv=10,scoring='f1');
printArray(ofObj,scores);
scores=cross_val_score(clf, x, y, cv=10,scoring=fpr_score);
printArray(ofObj,scores);
ofObj.write("Random␣Forest,");
clf=RandomForestClassifier();
scores=cross_val_score(clf, x, y, cv=10,scoring='f1');
printArray(ofObj,scores);
scores=cross_val_score(clf, x, y, cv=10,scoring=fpr_score);
printArray(ofObj,scores);
```

```
#main function
for fileName in fileNames:
    evaluate_classifier(fileName,outputFile)
    print ("Finished␣processing␣"+ fileName)
outputFile.close()
print ("!!!Experiement␣finished!!!!")
```

Appendix C

Python script for evaluation of classification time of candidate classifiers

```
import time
import numpy as np
from sklearn import preprocessing
from sklearn import tree
from sklearn.ensemble import AdaBoostClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.naive_bayes import GaussianNB
from sklearn import svm
print ("!!!Experiement started!!!!");
filePath="/home/biruk/workspace/DDoS expt/combined/20sec.csv";
data= np.genfromtxt(filePath,delimiter=',');
np.random.shuffle(data);#reorder the samples
train,test=np.vsplit(data,2);
ofObj=open('/home/biruk/workspace/DDoS expt/combined/processTime.csv','w'
    ↪ );
train_x= train[:,[0,1,2,3,4]];
train_y=train[:,5];
test_x= test[:,[0,1,2,3,4]];
test_y=test[:,5];
#Naive bays
ofObj.write("GNB,");
clf=GaussianNB();
_sum=0.0;
for x in range(0,10):
```

```
    clf.fit(train_x,train_y);
    start=time.clock();
    clf.predict(test_x);
    _sum+=(time.clock()-start);
ofObj.write(str(_sum/10.0)+"\n");
#DT
ofObj.write("DT,");
clf=tree.DecisionTreeClassifier();
_sum=0.0;
for x in range(0,10):
    clf.fit(train_x,train_y);
    start=time.clock();
    clf.predict(test_x);
    _sum+=(time.clock()-start);
ofObj.write(str(_sum/10.0)+"\n");
#Boost
ofObj.write("Boost,");
clf=AdaBoostClassifier(n_estimators=50);
_sum=0.0;
for x in range(0,10):
    clf.fit(train_x,train_y);
    start=time.clock();
    clf.predict(test_x);
    _sum+=(time.clock()-start);
ofObj.write(str(_sum/10.0)+"\n");
#SVML
xts=preprocessing.scale(train_x);
ofObj.write("SVML,");
clf=svm.SVC( kernel='linear');
```

```
_sum=0.0;
for x in range(0,10):
    clf.fit(xts,train_y);
    start=time.clock();
    xtt=preprocessing.scale(test_x);
    clf.predict(xtt);
    _sum+=(time.clock()-start);
ofObj.write(str(_sum/10.0)+"\n");
#SVMP
ofObj.write("SVMP,");
clf=svm.SVC( kernel='poly');
_sum=0.0;
for x in range(0,10):
    clf.fit(xts,train_y);
    start=time.clock();
    xtt=preprocessing.scale(test_x);
    clf.predict(xtt);
    _sum+=(time.clock()-start);
ofObj.write(str(_sum/10.0)+"\n");
#SVMR
ofObj.write("SVMR,");
clf=svm.SVC( kernel='rbf');
_sum=0.0;
for x in range(0,10):
    clf.fit(xts,train_y);
    start=time.clock();
    xtt=preprocessing.scale(test_x);
    clf.predict(xtt);
    _sum+=(time.clock()-start);
```

```
ofObj.write(str(_sum/10.0)+"\n");

#random forest

ofObj.write("forest,");

clf=RandomForestClassifier();

_sum=0.0;

for x in range(0,10):

    clf.fit(train_x,train_y);

    start=time.clock();

    clf.predict(test_x);

    _sum+=(time.clock()-start);

ofObj.write(str(_sum/10.0)+"\n");

ofObj.close();

print ("!!!Experiement␣finished!!!!");
```

Appendix D

Python script for evaluation of effect of session time on decision tree classifier

```
import numpy as np
from sklearn.metrics import make_scorer
from sklearn.model_selection import cross_val_score
from sklearn import preprocessing
from sklearn import tree
from sklearn.ensemble import AdaBoostClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.naive_bayes import GaussianNB
from sklearn import svm
filePath="/home/biruk/workspace/DDoS expt/combined/";
fileNames=["20sec.csv","40sec.csv","1min.csv","2min.csv","3min.csv","4min
    ↪ .csv","5min.csv","6min.csv","7min.csv","8min.csv"];
#open the output file
outputFile=open('/home/biruk/workspace/DDoS expt/combined/RQ2statistics.
    ↪ csv','w');
def false_positive_rate (truth,pred):
    FP = np.sum(np.logical_and(pred == 1, truth == 0));
    TN = np.sum(np.logical_and(pred == 0, truth == 0));
    return FP*1.0/(FP+TN)

fpr_score=make_scorer(false_positive_rate);#greater is better true
#function to print array content to a file
def printArray(outputFile,array):
    for x in np.nditer(array):
```

```
        outputFile.write(str(x)+",");
    outputFile.write("\n");
#classifier evaluation routine
def evaluate_classifier(fileName,ofObj):
    data= np.genfromtxt(filePath+fileName,delimiter=',');
    np.random.shuffle(data);#reorder the samples
    x=data[:,[0,1,2,3,4]]; #the feature matrix
    y=data[:,5]; #the target value as an array
    ofObj.write(fileName+"\n");
    ofObj.write("DT,");
    clf = tree.DecisionTreeClassifier();
    scores=cross_val_score(clf, x, y, cv=10,scoring='f1')
    printArray(ofObj,scores);
    scores=cross_val_score(clf, x, y, cv=10,scoring=fpr_score)
    printArray(ofObj,scores);
#main function
print ("!!!Experiement started!!!!")
for fileName in fileNames:
    evaluate_classifier(fileName,outputFile)
    print ("Finished processing "+ fileName)
outputFile.close()
print ("!!!Experiement finished!!!!")
```

Appendix E

Python script for evaluation of feature importance

```
import numpy as np
from sklearn.ensemble import RandomForestClassifier
filePath="/home/biruk/workspace/DDoS␣expt/combined/20sec.csv";
data= np.genfromtxt(filePath,delimiter=',');
np.random.shuffle(data);
x=data[:,[0,1,2,3,4]]; #the feature matrix
y=data[:,5]; #the target value as an array
clf=RandomForestClassifier();
clf.fit(x,y);
print clf.feature_importances_
```

Appendix F

ANOVA calculation of F1 score of decision tree, random forest and AdaBoost classifer

We used on-line one way ANOVA calculator available at **http://www.socscistatistics.com/tests/anova/default2.Aspx** for this ANOVA computation. The results are rounded for display purposes but no rounding was applied during computation. In the tables, X stands for individual F1 score, SS stands for sum of squares, df stands for degrees of freedom and MS stands for mean square.

Classifier	F1 score
DT	0.993503248,0.996494742,0.993,0.994497249,0.992488733, 0.995511222,0.992481203,0.99498998,0.99549324,0.995991984
Random Forest	0.992548435,0.996501749,0.993496748,0.995014955, 0.99500998,0.997503744,0.993496748,0.997005988,0.996505242,0.997
AdaBoost	0.994,0.996494742,0.992992993,0.995,0.993987976, 0.99550674,0.992978937,0.994491738,0.994491738,0.996494742

	DT	Random forest	AdaBoost
N	10	10	10
$\sum X$	9.9445	9.9541	9.9464
Mean	0.9944	0.9954	0.9946
$\sum X^2$	9.8892	9.9084	9.8932
Std.Dev.	0.0015	0.0018	0.0013

Result details

Source	SS	df	MS
Between F1 scores	0	2	0
Within F1 scores	0.0001	27	0
Total	0.0001	29	0

f-ratio=1.1339

p-value=0.336623

The difference between mean of the F1 scores is not statistically significant with 95% confidence.

Appendix G

ANOVA calculation of FPR score of decision tree, random forest and AdaBoost classifer

We used on-line one way ANOVA calculator available at **http://www.socscistatistics.com/tests/anova/default2.Aspx** for this ANOVA computation. The results are rounded for display purposes but no rounding was applied during computation. In the tables, X stands for individual FPR score, SS stands for sum of squares, df stands for degrees of freedom and MS stands for mean square.

Classifier	FPR
DT	0.007,0.002,0.007,0.005,0.005, 0.007,0.005,0.003,0.004,0.00
Random Forest	0.014,0.004,0.006,0.008,0.007, 0.004,0.006,0.005,0.005,0.003
AdaBoost	0.006,0.002,0.006,0.005,0.004, 0.006,0.003,0.004,0.003,0.002

	DT	Random forest	AdaBoost
N	10	10	10
$\sum X$	0.047	0.062	0.041
Mean	0.0047	0.0062	0.0041
$\sum X^2$	0.0003	0.0005	0.0002
Std.Dev.	0.0019	0.0031	0.0016

Result details

Source	SS	df	MS
Between FPR scores	0	2	0
Within FPR scores	0.0002	27	0
Total	0.0002	29	0

f-ratio=2.18465

p-value=0.132007

The difference between mean of the FPR scores is not statistically significant with 95% confidence.

Appendix H

ANOVA calculation of FPR score of SVML, SVMP and SVMR classifer

We used on-line one way ANOVA calculator available at **http://www.socscistatistics.com/tests/anova/default2.Aspx** for this ANOVA computation. The results are rounded for display purposes but no rounding was applied during computation. In the tables, X stands for individual F1 score, SS stands for sum of squares, df stands for degrees of freedom and MS stands for mean square.

Classifier	F1 score
SVML	0.977045908,0.981963928,0.970426065,0.975099602,0.976441103, 0.974384731,0.978894472,0.977977978,0.97995992,0.976558603
SVMP	0.978596317,0.981453634,0.970912738,0.974588939,0.975927783, 0.973895582,0.978402813,0.977466199,0.979469204,0.9750499
SVMR	0.978109453,0.981963928,0.969484742,0.975585451,0.9749499, 0.975438596,0.979407333,0.977977978,0.980490245,0.975585451

	SVML	SVMP	SVMR
N	10	10	10
$\sum X$	9.7688	9.7658	9.769
Mean	0.9769	0.9766	0.9769
$\sum X^2$	9.5429	9.5371	9.5434
Std.Dev.	0.0032	0.0031	0.0035

Result details

Source	SS	df	MS
Between F1 scores	0	2	0
Within F1 scores	0.0003	27	0
Total	0.0003	29	0

f-ratio=0.03013

p-value=0.970351

The difference between mean of the F1 scores is not statistically significant with 95% confidence.

Appendix I

ANOVA calculation of FPR score of decision tree classifier for different session time

We used on-line one way ANOVA calculator available at **http://www.socscistatistics.com/tests/anova/default2.Aspx** for this ANOVA computation. The results are rounded for display purposes but no rounding was applied during computation. In the tables, X stands for individual FPR score, SS stands for sum of squares, df stands for degrees of freedom and MS stands for mean square. for F ratio or F statistics

session time (seconds)	FPR
20	0.006,0.01,0.005,0.002,0.004, 0.002,0.006,0.006,0.007,0.007
40	0.004,0.004,0.005,0.007,0.004, 0.008,0.016,0.008,0.009,0.005
120	0,0.005,0.004,0.007,0.004, 0.003,0.005,0.009,0.001,0.003
240	0.007,0.007,0.003,0.004,0.004, 0.002,0.006,0.003,0.005,0.002
480	0.004,0.005,0.003,0.004,0.011, 0.002,0.003,0.004,0.003,0.003

	20	40	120	240	480
N	10	10	10	10	10
$\sum X$	0.055	0.07	0.041	0.043	0.042
Mean	0.0055	0.007	0.0041	0.0043	0.0042
$\sum X^2$	0.0004	0.0006	0.0002	0.0002	0.0002
Std.Dev.	0.0024	0.0037	0.0026	0.0019	0.0025

Result details

Source	SS	df	MS
Between FPR scores	0.0001	4	0
Within FPR scores	0.0003	45	0
Total	0.0004	49	0

f-ratio=2.12825

p-value=0.092844

The difference between mean of the FPR scores is not statistically significant with 95% confidence.