Thesis Dissertation

EVALUATION AND ANALYSIS OF A COLLABORATIVE MISBEHAVIOUR MODEL

Nikolas Venizelou

UNIVERSITY OF CYPRUS

COMPUTER SCIENCE DEPARTMENT

May 2023

UNIVERSITY OF CYPRUS
COMPUTER SCIENCE DEPARTMENT

Evaluation and analysis of a collaborative misbehaviour model

Nikolas Venizelou

Supervisor
Dr. Vasos Vasilliou

Thesis submitted in partial fulfilment of the requirements for the award of degree of Bachelor in Computer Science at University of Cyprus

May 2023

Acknowledgments

I would like to thank my supervisor Dr. Vasos Vasiliou for providing me such an interesting thesis project and guiding me throughout the problems that I faced during my research, but most importantly I am really glad for our back and forth conversations during the thesis meeting where he never failed to listed to my opinion and thinking process.

Moreover, I need to thank Dr Marios Thoma who was kind enough to provide my with his implementation of the model as well as answer questions that came up when I trying to understand his PhD thesis.

Additionally, I would like to thank the Computer Science department, especially the teaching staff that managed to pass me valuable knowledge will also sharing their passion for computer science.

Finally I want to thank my family for supporting me throughout the years and my friends with whom I cherish valuable memories created in the last couple of years while attending this University.

Abstract

Distributed Denial of Service attacks is the major thread on the availability of resources in computer networks for at least the last 20 years, due the the challenging problem of distinguish such attacks, as attackers continue to evolve with detection mechanisms and defence techniques. This thesis aims to evaluate and analyze a model for detection of collaborative misbehaviour users in DDoS attacks, that detects early signs of abnormal user behaviour based on the frequency of the users requests in correlation with time. This document will explain why the successful identification of collaborative misbehaving users can be used to detect early sign of DDoS attacks and the underline method used in the original paper. Furthermore, a new implementation of the model written in python will also be provided and explain the reason for creating such implementation even though the original model's code was kindly provided by the author.

Most importantly this document will analyse the selection of a time interval for which the users requests are observed and measured in order to determine if users are collaborators, by selecting and measuring the accuracy of the original program, using different time intervals.

This thesis argues that the selection of a correct time interval has the most significant role on accurately detecting collaborative users of the model, and also with the use of real-world data result measurements and other logical realizations, questions the accuracy and correctness of the model's algorithm. This thesis concludes that the simple logic proposed in the original paper has overlooked many factors which in turn are observable by deeply analysing the model.

Contents

List of Figures

List of Tables

List of Algorithms

Chapter 1

Introduction

1.1 Main idea and contribution

This thesis extends and is based on the previous work done on detecting collaborative misbehaviour in DDoS attacks [11], where the paper focuses on identifying collaborators in a network by observing network traffic and the time interval between each user request. This results in a systematic method of collecting valuable meta-data about the network traffic in general which is easily applied to detect malicious users or collaborators due to the nature of DDoS attacks. In practice the model determines if a user is a collaborator with another user, by finding users that issue requests approximately simultaneously based on the relation between the user's expected frequency (of requests) and measured frequency with a mathematical approach based on the theory of hidden Markov models.

This thesis purpose is to re-create a clean and understandable implementation of the model in a modern programming language as well as analyse and evaluate the model with the possibility of proposing improvements to the underline methodology described in the original paper.

More specifically this study aims to find if a time interval, to measuring the frequency of users requests, exists for which there are improvements in the detection rate and accuracy. To achieve this an implementation of the method was provided by the author of the original paper and with some modification results were taken to demonstrate this the difference between time intervals, by using a real botnet data. Furthermore, proposed improvements will be mentioned in the practical implementation of the model as well as the ability of integrating this model to existing infrastructures will be questioned.

1.2 Thesis structure

The structure consists of a total of 7 sections, in section 2 the collaborative misbehaviour model is further explained with a subsection providing a basic understanding of the hidden Markov model theory and how it used in the model, as well as a general background in DDoS attacks and how an attacker might deploy such attacks. Furthermore, section 3 will provide a python implementation of the model, as well as the practical implementation reasoning which will help to understand the detection mechanism as well as some of its shortcoming. Continuing, section 4 provides an analysis on the time interval selection, using the implementation and the CAIDA DDoS attack dataset [1]. Section 5 includes the evaluation of the model, which explains the uniqueness of the model in contrast to other detection mechanisms, and also explains the problems found in the logic of the proposed algorithm while mentioning ways of improvement. Finally, section 6 describes the future work that could be done and section 7 concludes this thesis.

Chapter 2

Background

2.1 General

In order to understand how the collaborative misbehaviour model's [11] time interval selection can be a major factor in the model's detection and accuracy, how the model can be compared with other detection techniques and how it can be implemented to coexists with existing infrastructures, one must first understand the model itself and also the mathematical reasoning used to create the model. General knowledge on DDoS attacks and variations is also expected, to provide the basic knowledge on understanding the detection itself.

2.2 DDoS attacks

Distributed Denial of Service (DDoS) attacks are a major threat to many organizations, ranging from small businesses to large corporations and government agencies. These attacks can cause significant disruption to an organization's online services, resulting in lost revenue, damage to reputation, and other negative consequences. DDoS attacks are targeting the availability of resources by overwhelming a network with useless traffic and ultimately prevent legitimate users from accessing the service. In a DoS attack, a single computer or network is used to flood the targeted site or network with an excessive amount of traffic, causing it to crash or become unavailable. A DDoS attack is similar, but instead uses a large network of computers, often controlled by a botnet, to flood the targeted service with traffic from multiple sources. The severity of the attack is dependent on the amount and type of traffic generated, and the defenses in place to mitigate it. Both DDoS and DoS attacks can have significant impacts on the targeted service, such as lost revenue, damaged reputation, and disrupted operations. These attacks can be initiated by botnets and other compromised devices, making it challenging to identify and mitigate them, in

this section we take a look at the different types of DDoS attacks and their characteristics.

2.2.1 DDoS in OSI Layers

DDoS attacks can occur in multiple network layers of the Open Systems Interconnection (OSI) model, namely the network (Layer 3), transport (Layer 4), and application (Layer 7) layers. In the network layer, DDoS attacks typically exploit protocols such as the Internet Protocol (IP) and the Internet Control Message Protocol (ICMP). These attacks involve overwhelming the target's network resources with a high volume of traffic or exploiting IP fragmentation vulnerabilities. In the transport layer, attacks focus on protocols such as the Transmission Control Protocol (TCP) or the User Datagram Protocol (UDP) [6], aiming to exhaust server resources or disrupt the communication between network hosts. Application layer attacks occur at Layer 7 and target vulnerabilities specific to the application itself, such as the Hypertext Transfer Protocol (HTTP) or the Domain Name System (DNS) [8].

Effective detection and defense techniques vary across network layers. In the network layer, common techniques include traffic anomaly detection, source IP filtering, or implementing routers with access control lists to filter unwanted traffic. In the transport layer, techniques such as rate limiting, SYN cookies, or connection tracking can help identify and mitigate attacks [6]. At the application layer, specialized web application firewalls (WAFs) and behavior-based anomaly detection are commonly used to detect and block malicious traffic [8].

Each layer attack differs in terms of the specific vulnerabilities they exploit and the impact on the targeted system. Network layer attacks primarily focus on overwhelming network resources, causing congestion and disrupting normal network operations. Transport layer attacks aim to exhaust server resources by exploiting vulnerabilities in connection establishment or teardown processes. Application layer attacks directly target the functionality or resources of the application, impacting the service's availability or performance. While the effectiveness of each attack may vary depending on the target and the specific attack technique employed, application layer attacks are often considered more effective as they can directly affect the desired service or functionality.

2.2.2 DDoS Techniques

Distributed Denial of Service (DDoS) attacks employ various techniques in addition to selecting one or multiple layer to attack. These techniques are often employed by attackers to exploit vulnerabilities and evade detection, making defense more challenging.

One prevalent technique used by attackers is the utilization of botnets. Botnets consist of a network of compromised devices, or bots, under the control of an attacker. By coordinating the actions of these bots, attackers can generate a massive volume of traffic directed towards the target. The distributed nature of botnets makes it difficult to pinpoint the source of the attack and mitigating its impact, as the attack traffic appears to originate from multiple sources. Furthermore, the attacker can issue commands to the botnet, controlling and adjusting the attack strategy as needed. This technique enables attackers to overwhelm the target's resources, such as bandwidth, processing power, or memory, leading to service disruption or unavailability [4].

Another technique employed by attackers is low-rate attacks. These attacks are characterized by their slow and subtle consumption of the target's resources over an extended period. By maintaining a low traffic rate, attackers aim to evade detection mechanisms that rely on monitoring for sudden spikes in traffic. Instead, they gradually exhaust the target's resources, making it challenging to differentiate between legitimate and malicious traffic. The goal of low-rate attacks is to cause a long-term degradation of the target's performance, often resulting in a complete denial of service over time [9].

Reflective attacks represent another commonly used technique by attackers. These attacks take advantage of vulnerable network protocols that can be abused to amplify the volume of attack traffic. Attackers send requests to a large number of reflective servers, forging the source IP address to that of the target. The reflective servers, unaware of the forged source, respond by sending their replies to the target. Due to the amplification effect, the responses from the reflective servers are much larger than the initial request, resulting in an overwhelming flood of traffic towards the target. Reflective attacks enable attackers to maximize the impact of their attacks using fewer resources and amplifying the traffic volume by exploiting vulnerable protocols [3].

In addition to these attack techniques, attackers employ various evasion tactics to avoid detection and hinder defense efforts. One such tactic is IP spoofing, where the source IP address is forged to make it appear as if the traffic originates from legitimate sources. This technique makes it difficult to trace the origin of the attack, as the source IP address does not accurately represent the attacker's identity. Defenders face the challenge of distinguishing legitimate traffic from spoofed traffic, complicating the detection and mitigation process.

2.2.3 DDoS Ramp-up strategy

Ramp-up is a unique prominent strategy deployed by the majority of today's Distributed Denial of Service attacks. DDoS attack ramp up involves a carefully orchestrated escalation of attack intensity over time. Instead of overwhelming the target instantly, attackers gradually increase the attack's scale, duration, or diversity of attack vectors. This incremental approach exploits the thresholds and limitations of target systems, effectively overwhelming their infrastructure and extending the disruptive effects. By deploying various attack vectors in a staggered manner, such as volumetric, protocol, or application-layer attacks, perpetrators heighten the chances of circumventing defensive measures and further prolonging the attack's impact. The adoption of DDoS attack ramp up stems from several motivations. Initially launching a milder attack allows attackers to evade early detection and mitigation efforts employed by target organizations. This strategic approach enables probing the resilience of defenses, adapting attack vectors accordingly, and capitalizing on any identified weaknesses during the ramp-up phase. The gradual nature of the attack also serves to assess the target's response capability, potentially inducing fear in the victim and increasing the likelihood of achieving intended objectives, such as extortion, service disruption, or gaining competitive advantage [2].

2.3 Collaborative misbehaviour model

In this section a background about the methodology of the collaborative misbehaviour model will be provided as well as an explanation of the underline mathimatical reasoning of Hidden markov models in simple terms in-order to grasp the practical implementation if this method.

2.3.1 Hidden markov models

A discrete time Markov chain is a mathematical model used to describe a system that changes over time and transitions between a finite set of states. It follows the Markovian property, which states that the probability of transitioning to a particular state at the next time step only depends on the current state.

In this model, we have a set of states Θ and a transition probability matrix P that determines the likelihood of moving from one state to another. At the beginning, there is an initial probability distribution π_0 that represents the chances of starting in each state. As time progresses, the probabilities of being in different states can be calculated iteratively using the transition probability matrix and the previous probabilities.

The Markovian property states as an equation

$$\begin{aligned} Pr(X_n = x_{i_n} | X_0 = x_{i_0}, \ldots, X_{n-1} = x_{i_{n-1}}) \\ = Pr(X_n = x_{i_n} | X_{n-1} = x_{i_{n-1}}), \end{aligned} \tag{2.1}$$

where Pr

If the transition probabilities do not change over time (homogeneous Markov chain), the probabilities can be calculated simply by multiplying the previous probabilities with the transition probability matrix. This matrix is column stochastic, meaning that the probabilities for transitioning from each state to all other states sum up to 1.

A hidden Markov model (HMM) extends the concept of a Markov chain by introducing the idea of observed outputs. In addition to the states and transition probabilities, an HMM includes a set of possible output symbols and an emission probability matrix. This matrix determines the probability of observing a particular symbol given the current state. Each state is associated with a specific output symbol.

The HMM is useful for modeling systems where we have observable outputs that are influenced by underlying states, but we cannot directly observe the states themselves. By analyzing the observed outputs and using the HMM, we can make inferences about the hidden states and their transitions.

In summary, a discrete time Markov chain is a model that represents how a system transitions between states over time. The hidden Markov model extends this concept by incorporating observed outputs and the probability of emitting those outputs from each state. These models are widely used in various fields, including speech recognition, bioinformatics, and natural language processing.

2.3.2 Approach

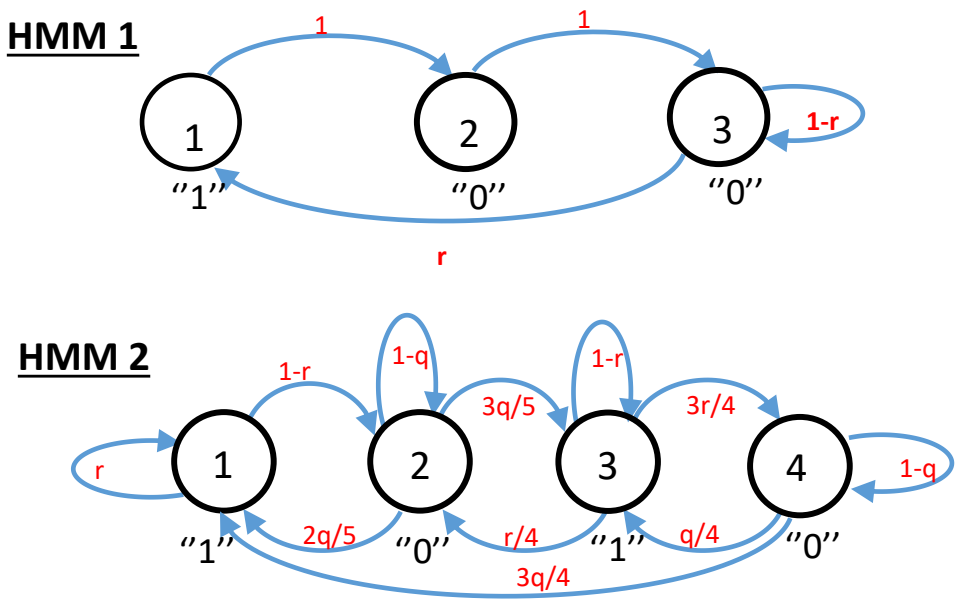

Figure 2.1: HMM1 and HMM2

The paper [10] concluded that using the 2 hidden Markov models shown above, correlation between users can be identified by observing the time a user has sent a packet or not (indicated by a 1 or a 0) in order to come identify users or a group of users as collaborators in a network. In order for this to happen, a sufficient N must be selected which represent the time slots for which we check if a user has sent data to the network or not. This is a really important part of the detection model and essentially the main question we want to answer in this paper is that if there is such a value for N for which we can have the best accuracy, and if such a value exists if it is bound to specific network traffic or not.

The value N that was previously mentioned will be refer to as t_s, which is defined as the time slot for which determine if a user has send a packet or not, indicating 1 or 0 respectively. If T_s is used to denote the total length of the inspection window (for example the length of the dataset, measured in time) we have that:

$$t_s = \frac{T_s}{n} \tag{2.2}$$

Where n denotes the total number of timesteps. By using hidden markov models, linear algebra and other mathematical reasoning the original paper [11] proved the following: lets assume that we have a MxN matrix where M is the number of users (unique IPs) and N is the total number of timesteps (n described on the previous equation), denote that each row of the matrix can be considered as $y[k] \in 0,1,\ \forall k \geq 0$ and represent the activity of a user. For some user 1 and some user 2 we have

- For user 1, let $\hat{u}_1$ the empirical frequency where user 1 sends a packet (number of terms where we have $y_1[k] = 1$) :

$$\hat{u}_1 = \frac{number \quad of \quad terms \quad where \quad y_1[k] = 1}{n} .$$

- For user 2, let $\hat{u}_2$ the empirical frequency where user 2 sends a packet (number of terms where we have $y_2[k] = 1$) :

$$\hat{u}_2 = \frac{number \quad of \quad terms \quad where \quad y_2[k] = 1}{n} .$$

- For both, let $\hat{u}$ the **comparable frequency** for which both user1 and user2 send packets at the same timeslot k :

$$\hat{u} = \frac{number \quad of \quad terms \quad where \quad y_1[k] = y_2[k] = 1}{n} .$$

- We also have the **relative frequency** $\hat{u}_{12}$ which corisponds to the product of $\hat{u}_1$ $\hat{u}_2$:

$$\hat{u}_{12} = \hat{u}_1 \hat{u}_2$$

By comparing the comparable frequency and relative frequency of 2 users we can understand if the users are collaborative or not. If 2 users are independent (non-collaborative) then the relative frequency and comparable frequency should be close to equal $\hat{u}_{12} \simeq \hat{u}$ for $n \to \infty$.

In the original article, it is emphasized that the knowledge of Hidden Markov Models (HMMs) for modeling user behavior is not necessary for the implementation of the proposed approach. The primary focus of the analysis lies in the utilization of the only observable information available to us, namely the network requests made by users, in order to detect any anomalous activities. This is achieved by calculating the discrepancy between the relative frequency and the comparable frequency, thereby serving as an indicator of the correlation degree between the requests. It should be noted that the exact quantification of the correlation between users, in terms of specific values, is not explored within the scope of the present study. Additionally, the authors suggest the exploration of correlations over time periods by applying the same methodology to shifted versions of the sequences

Chapter 3

Implementation

In this section we will take a look at the practical implementation of the model, which has a complexity of $O(n^2 * m)$, with n and m denoting the number of users and timesteps respectively. The implementation was provided by the author of the original paper, and some improvements and suggestions for the advancement of this model will be included in this section. It is important to note, that even though hidden Markov models were an important part in the original research, in-order to provide a mathematical model of the collaborative user detection problem, it it not relevant beyond the point of proving that by observing the comparable and relative frequency of users we can understand it the users are in collaboration. Thus, the implementation does not contain any programmatic implementation of hidden Markov models nor is their a reason to put such an effort on implementing them. Continuing, this section we will describe the pseudo-algorithm used in the implementation, an abstract view of the implementation itself, how the network traffic is being processed as well as some limitation of the current implementation.

3.1 Algorithm - Logic

Users / Timesteps	1	2	3	4	5	6	7	8	9	10
IP 1	1	0	1	0	1	0	1	0	1	0
IP 2	0	1	0	1	0	0	1	0	1	1
IP 3	0	0	1	0	0	1	0	0	1	0
IP 4	1	1	1	1	1	1	1	1	1	1
IP 5	1	1	1	1	0	0	0	0	0	0
IP 6	0	0	0	0	1	1	1	0	1	0

Table 3.1: Timestep matrix

Assuming that we have a matrix like 3.1 where each column represents a timestep from 1 to n (recall n from equation 2.2, and the rows a unique IP in the network traffic. Every time a columns of matrix has a value of one, it is indicated that the IP of that row has network activity for the given timestep. By creating such a matrix for a network for which we want to examine if collaborators in the network exist, the original paper proposed the following algorithm (the algorithm removes some unnecessary complexity in-order to simplify the logic and appear more understandable while still produce the same results).

The algorithm 1 shows the basic function form of the proposed algorithm with some modifications. The algorithm focuses on creating and comparing the frequency of activity of each user by using the equations described in section 2.3.2, this frequency are later used to identify depended (collaborative) and independent (non-collaborative) users, simple by comparing their values.

The algorithm assumes that we have a matrix as described in table 3.1, as well as an array with the unique IP addresses of the users that are active in the network at the time of the analysis (or in the corresponding data-set/network traffic PCAP).

With the two nested for loops shown in the algorithm, we can make all comparisons between pairs of unique IPs.

For each pair, we compute the probability $\hat{u_1}$ (corresponding to prob_ip1), $\hat{u_2}$ (corresponding to prob_ip2), and the comparable frequency/probability $\hat{u}$ (corresponding to comparable). After calculating the necessary probabilities, we then compare the comparable frequency with the relative (prob_ip1 $\times$ prob_ip2) in order to identify the pair as collaborators. For every identification of a user pair a counter is used (dep and $indep$)of the times each user was found depended and independent respectively.

After a user is compared with all other users (after the end of the nested loop j) the user i is added in the depended or independent set by comparing $dep[i] \geq indep[i]$ or $dep[i] < indep[i]$. An important criterion that is also added is the identification of the pair as collaborators in the case of $\hat{u}_1 = \hat{u}_2 = \hat{u}$, which ultimately means that the pair has exactly the same activity, in this case the counters are not used for the identification but rather the pair is automatically added to the depended set and assumed to be collaborators.

Algorithm 1 Detection model main algorithm

```
function ComputeProbabilities(matrix, unique_ips)
    num_ips ← length(unique_ips)
    num_columns ← length(matrix[0])
    depended ← {}
    independent ← {}
    dep ← length(unique_ips)
    indep ← length(unique_ips)
    for i ← 0 to num_ips − 1 do
        ip1 ← unique_ips[i]
        prob_ip1 ← count_ip1 / num_columns
        count_ip1 ← number of times ip1 has 1 in a timestep
        if ip1 ∈ depended then
            independent.add(ip1)
        for j ← i + 1 to num_ips − 1 do
            ip2 ← unique_ips[j]
            count_ip2 ← number of times ip2 has 1 in a timestep
            count_both ← number of time ip1 = ip2 = 1 in a timestep
            prob_ip2 ← count_ip2 / num_columns
            comparable ← count_both / num_columns
            relative ← (prob_ip1) · (prob_ip2)
            if prob_ip1 == prob_both and prob_ip2 == comparable then
                depended.add(ip1)
                depended.add(ip2)
                continue
            else if comparable > relative then
                dep[i]++
                dep[j]++
            else
                indep[i]++
                indep[j]++
        if ip1 ∈ depended then
            continue
        else if indep[i] > dep[i] then
            independent.add(ip1)
        else
            depended.add(ip1)
    return
```

The author of the original paper also included detection for other groups of users which seem to be irrelevant or inaccurate as will be seen in the analysis section. The total groups identification in the original algorithm included:

- **Depended users**: Which are users that we consider to be in collaboration with either one or multiple users. This is determined as already mentioned by comparing the relative and comparable frequency $F_{comparible}$ (same as $\hat{u}$) > $F_{relative}$ (same as $\hat{u}_{12}$). This comparison is different from the original $\hat{u}_{12} \simeq \hat{u}$ since in the practical implementation of this method we do not have $n \to \infty$

- **Independent Users**: Similarly no collaborative users are also detected with the exception that the comparison is $F_{comparible} \leq F_{relative}$

- **Abnormal Users**: Are considered users with abnormally high activity with the proposal identification of such users being that they have observed packet rate higher that 0.9 (with 1 being the maximum). This essentially means that in the case we have 10 timesteps, if the user shows activity in at-least 9 of them, the it is considered an abnormal user

- **No-activity Users**: Are easily described as users that have 0 activity through-out the analysis / detection model

- **Flash-crowd**: Considered as the most problematic user category, a flash crowd can be described as a sudden surge in legitimate user traffic towards a specific web resource or service and will be explain in more detail in section 3.2, for now, these users are identified by having a packet rate higher that the mean packet rate.

The flowchart diagram, figure 3.1, shows a logic of the algorithm 1 will also adding identification for flash crowd and abnormal users with the criteria described by the author. As already mentioned, we found it was at best interest to consider group identification beyond depended and independent users (collaborative / non-collaborative) out of scope in the new implementation, the reason being that other groups were more prone to produce inaccurate results due to the following reasons:

- No-activity users: This is consider irrelevant due to the simple reason that if a user have no activity it will simply not appear in the data-set or pcap file used for the detection.

- Abnormal users: Generally abnormal traffic is considered traffic that differentiate from normal operation traffic, such as DDoS attacks, port scanning, botnet activities. In the original paper, abnormal users are considers users with packet rate higher than 0.9 (with 1 being the case of having activity on every timestep), with

the every organization deploying rate limiting on network users, detecting for abnormal users or rather detecting high traffic users, is a very basic and out-of-scope identification for this detection model as we should sorely focus on the detection of collaborators rather than create a "jack of all trades" DDoS detection model as there already many papers that focus entirely on abnormal traffic detection [12].

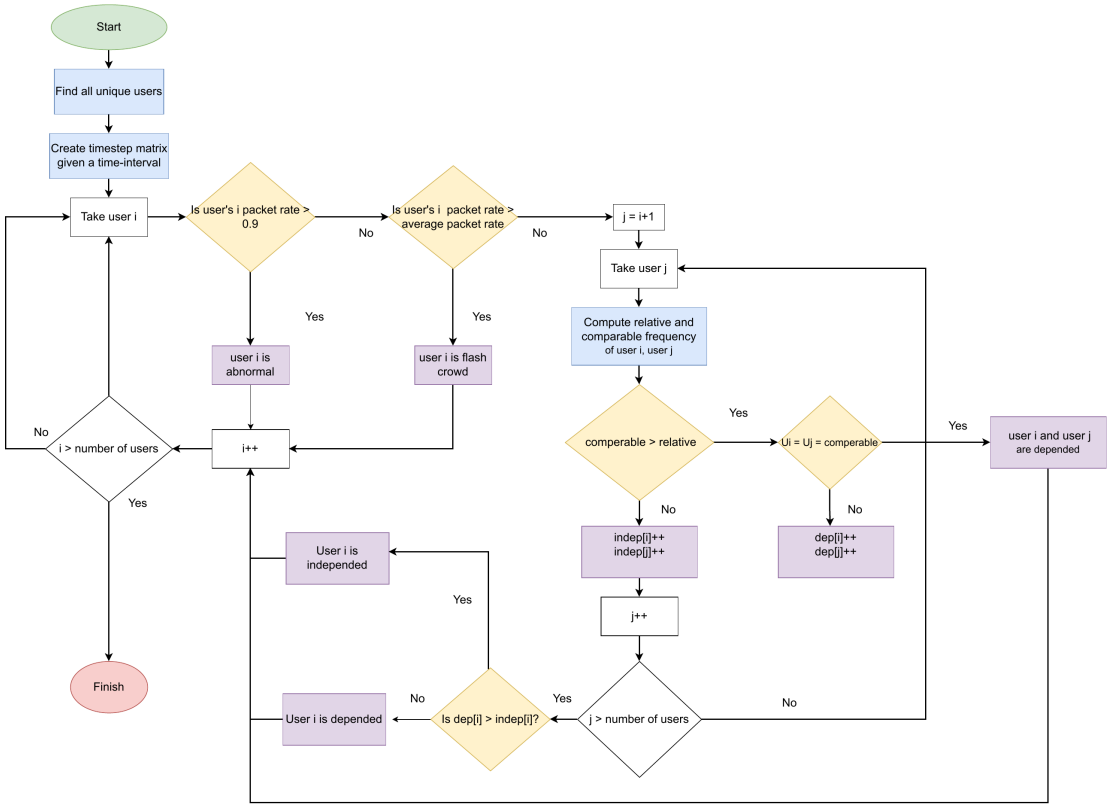

Figure 3.1: Algorithm Flowchart

3.1.1 Flash Crowd Detection

As already mentioned flash crowds pose significant challenges in DDoS detection due to their unique characteristics and the difficulties in differentiating them from actual DDoS traffic. A flash crowd refers to a sudden surge in legitimate user traffic to a particular website or online service, typically triggered by a popular event or viral content. From a DDoS detection perspective, this category of users is considered the most problematic.

The primary reason for the challenge lies in the similarities between flash crowds and DDoS attacks. Both scenarios involve a sudden increase in incoming traffic, which can overwhelm the target infrastructure and disrupt normal operations. This similarity makes it inherently difficult to distinguish between benign flash crowd traffic and malicious DDoS traffic based solely on traffic volume or patterns.

Moreover, flash crowds intensify the detection challenge by exhibiting high degrees of similarity to DDoS attacks in terms of traffic characteristics, such as packet rate, payload size, and distribution. This similarity further complicates the task of accurately identifying and mitigating DDoS attacks, as it becomes crucial to differentiate between legitimate traffic from interested users and the coordinated malicious traffic of a DDoS attack.

The original paper's approach of detecting flash crowds, is simply comparing the user's packet rate with the average packet rate, by having a higher packet rate than the average the user is then consider part of the flash crowd group. What it is believe to be overlooked is that is not a characteristic that is unique to flash crowd users but rather it is valid for both DDoS / DoS users, meaning in a normal network traffic having higher than normal packet rate means that the source is either part of a DDoS attack or part of a flash crowd. Inherently this simple criteria does not solve the bigger problem of distinguishing DDoS attacks from flash crowds and it can be easily proven by observing the results of the original implementation, where even though no flash crowd traffic was present, for logical time-interval selection it was almost certain that a user will be identified as flash crowd. In the original paper, it was determined that a flash crowd user is most certainly a depended user (can be seen in appendix 1), but the interpretation may overlooked the important aspect that in contrast to botnet controlled DDoS attacks, flash crowds are not necessarily deployed in a collaborative manner, since each user might be active in an entirely different timestep. The key factor that is overlooked is that flash crowds are much more harder to identify that simply comparing the mean packet rate and is something that could be researched further by incorporating probability metrics detection described in [5]

3.2 Implementation analysis

In this subsection we will provide the motivation for re-implementing the collaborative misbehaviour model, the differences between the 2 models including in terms of complexity and efficiency. Finally a prove of correctness will be provided that will compare the results from both the models in-order to make sure that no modifications were made that could affect the results and conclusions of this paper. It also worth mentioning that even though the new implementation will be used in the later sections for analysing the model, the original implementation will also be considered since future section will compare the results taken from the original paper [11].

Any implementation of this model, will ultimately need to implement some basic operations in-order to create the logic of the model. The main logic can be broken down into 4 major section:

1. Identifying unique users
2. Selection of time-interval and calculation of total timesteps
3. Creating the timestep matrix like table 3.1
4. Calculating probabilities and differentiate user groups

3.2.1 High-level idea of original implementation

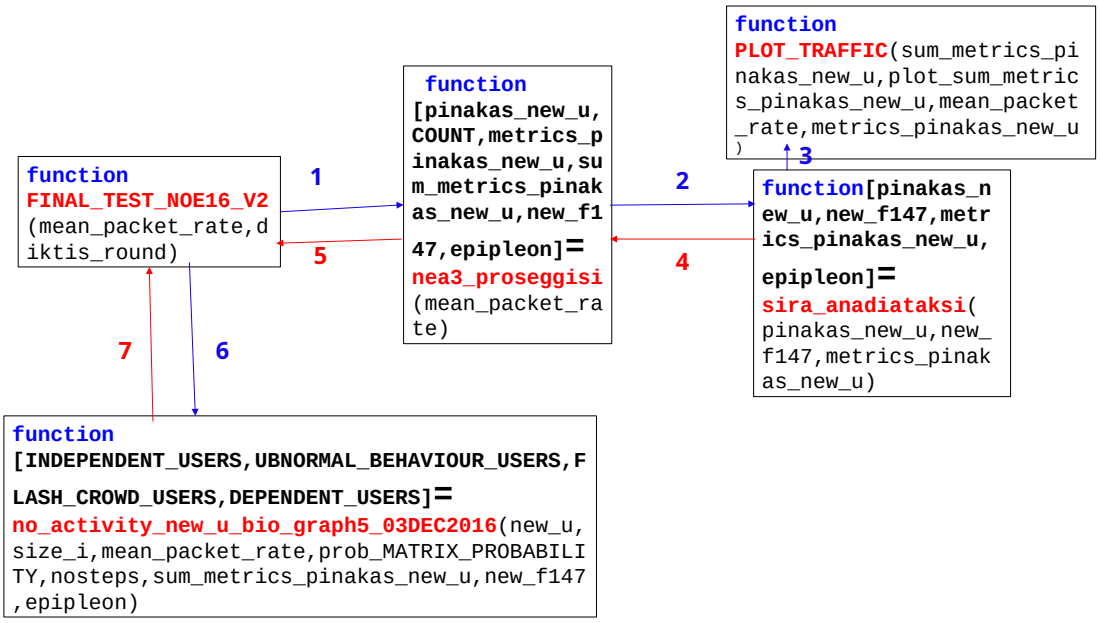

Figure 3.2: Original Implementation diagram

The original implementation's diagram can be seen on figure 3.1 where it is important to mention that the original model was implemented in Matlab (an expressive proprietary multi-paradigm programming language) as it is a simple enough to learn, offer data visualization (for debugging and understanding the programming logic) and also provides easy interface for creating charts. As it can be seen from figure 3.1 the implementation is broken down into 5 different files, where each file can be seen as a different function or method.

The control flow of the program goes as follows, with FINAL_TEST_NOE16_V2 referred as the main function:

1. The main function takes the input file containing the dataset and calls "nea3_proseggisi" function

2. "nea3_proseggisi" takes the dataset and with the help of "sira_anadiataksi" function creates the matrix described in table 3.1

3. Later with the use of "PLOT_TRAFFIC" the traffic is plotted and also calculates the mean packet rate and user mean packet rate

4. Control is transfered back to the main function were the "no_activity..." function is called

5. "no_activity..." creates a matrix where it contains probabilities and if the user is part of abnormal, no-activity or flash crowd group. It also creates a graph with the relationships between users and finally prints the percentage of each group.

It is worth noting that the whole Matlab program come in total of 1300 lines of code.

3.2.2 Motivation for implementation - Differences

There are many reasons for the re-implementation of the model, but most importantly it is worth noting the original model was model meant to be used to prove the theoretical work of Dr. Marios Thoma in the paper [11] was working and to provide some observations and parts that could be improved with the underline methodology and variable selection in future works. A logical approach was to implement the model in a more modern and functional programming language in-order to re-write it in a more clean and straight forward manner while also improving some bugs that were found in the original implementation.

Some important bugs that were found were:

- The matrix that was scanned from the data-set files was mapped incorrectly in terms of data types in the program's matrix. This resulted in IP addresses represented by a double value instead of a string, which had a big effect on the overall detection accuracy since the unique IP addresses were not identified correctly.
- The program did not used the user's IP as identifier for the users, making it hard to understand the results of the method since the identifiers used had no correlation with the actual data-set. This made the analysis of the model extremely difficult since no observation could be made about the false positives / false negatives without modifying the code
- Finally, the code responsible for producing the timestep matrix produced invalid results with users appearing to be inactive when that was in fact impossible since a user must exist in the data-set in-order to be included in the timestep matrix.

Overall the program was understandably hard to read, improve and fix, since it went through many modification in the original research process. This made the implementation that was receive include unnecessary code and complexity but also many coding blocks that were not used at all in the methodology, creating the motivation for implementing the model.

3.2.3 High-level idea of new implementation

As already mentioned the new implementation is written in python 3 with one of its goals, being to write an easily readable and understandable code, that can be encapsulate the detection's logic in a simple program. The new implementation also fixed the bugs that were found in the original implementation which in turn, deemed the experimental results found on the original paper as invalid, since the data-set parsing of the original implementation produced a different timestep table both in terms of users and timestep activity as mentioned on the previous section.

The new implementation comes to a total of 120 lines of code, and it is measured to be at least 5 times faster that the original implementation, possibly due to the use of more efficient data structures such as sets and lists. It important to mention that this implementation does not in fact identifies abnormal, no-activity or flash crowd users as it was already explained why it is believed that the proposed criteria for such identification is considered unnecessary/out-of-scope or inaccurate on section 3.1. The probability calculation and user identification of this implementation explicit follows the algorithm described rather than the flowchart and produces the following results:

- The total number of unique users
- The total number of timesteps for the selected time-interval that is taken as an arguments
- The percentage of depended and independent users
- Depended and Independent set of users that are identified by using IP

3.2.4 Proof of correctness

User/timestep	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39	40
User1	1	0	0	0	1	0	0	0	1	0	0	0	1	0	0	0	1	0	0	0	1	0	0	0	1	0	0	0	1	0	0	0	1	0	0	0	1	0	0	0
User2	1	0	0	0	1	0	0	0	1	0	0	0	1	0	0	0	1	0	0	0	1	0	0	0	1	0	0	0	1	0	0	0	1	0	0	0	1	0	0	0
User3	1	1	0	0	0	1	0	0	0	1	0	0	0	1	0	0	0	1	0	0	0	1	0	0	0	1	0	0	0	1	0	0	0	1	0	0	0	1	0	0
User4	1	0	1	0	0	0	1	0	0	0	1	0	0	0	1	0	0	0	1	0	0	0	1	0	0	0	1	0	0	0	1	0	0	0	1	0	0	0	1	0
User5	0	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1	0	0	0
User6	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1	0	0	0	0
User7	1	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0

Figure 3.3: Correctness timestep matrix

In order to prove that the implementation that was made for this thesis produces the same results with the original implementation, consider the figure 3.3 which represents the timesteps matrix for an example network activity of 7 users. The rows of the table represent the unique source (user) and the columns represent the timesteps which are 40 in total, each unique color of the boxes containing a 1 digit shows that the these users share activity on the selected timestep (column). From the previous section that we explain how the implementation of the method works, we can observe that user 1 and user 2 are definitely collaborative.

The results of the implementations of the model were both the same and concluded that all users except user 3 and user 4 are collaborative. It is important to note that this small test is not focuses on the accuracy of the model but rather to proof that both implementations produce the same outputs.

The outcome of the both implementations were exactly the same, since the bugs of the original implementation were found on the parsing of the data-set and creation of the timestep matrix, where in this case was already provided as an input. By this test we assume that the detection model behaves and results the same in both implementation, with the new implementation being proven to sufficient to be used in the analysis of the model.

Chapter 4

Analysis

This section focuses on the analysing the model in terms of accuracy and correctness. More specifically in this section we focus on providing an analysis of the results that the model produces using real-world DDoS attack data-sets, but also we will be testing different time-intervals in-order establish if a time-interval value exists for which we have the most accurate results. The sections will start by examining the results taken from the correctness results of the implementation section and then later will continue by examining the CAIDA data-set on both low-rate and high-rate DDoS attacks. Finishing comparison of the results with the original implementation will be provided as well as some final observations about the analysis itsself.

4.1 Base line - Simple test

As already mentioned we will try to explain the results taken from section 3.2.4 and also provide a first test / analysis of the model. The model was run using the timestep matrix shown on figure 3.3 and the results were the following:

	Total users	**Total timesteps**	**Collaborators (users %)**	**Non-Collaborators (users %)**
Results	7	40	71.43	28.57

Table 4.1: Simple test results

The results concluded that from the 7 users that participated in the detection model, 5 of them were collaborative. To first glance the results appear as simple incorrect since only 2 of the users show on the timestep table are undeniably collaborators (user 1 and user 2) although, many of the other users seem also have common network activity with a small time-shift. The explanation for the behaviour described is simply that the model

was not designed to be run on such a small amount of timesteps, due to the fact that with small timesteps the basic reasoning of the model is compromised since there is a higher chance for a user be active at a same timestep which another user. This is in agreement with the observation made from the original author that in-order to get reliable results from the model we must have a timestep number higher than $10^3 - 10^4$

4.2 Caida Dataset Testing

This analysis used the widely evaluated CAIDA dataset [1], which includes a total of 13 files containing 5-minute duration (equivalent to 300 seconds) of anonymized network traffic from a DDoS attack that occurred on August 4, 2007. These data-sets specifically captures the traffic related to the attack, including both the attack traffic directed at the victim and the corresponding response from the victim. Non-attack traffic has been filtered out to the greatest extent possible. According to a study by Moore [7], an attack is considered high-rate if it involves more than 10,000 packets per second across the network. On the other hand, if the attack involves 1,000 attack packets per second, accounting for 60% of the total attack traffic, it is categorized as a low-rate attack. In this subsection we will test different time-intervals for different data-sets of the CAIDA data-set family, broken down into 2 categories : Low-rate and high rate in accordance to Moore [7]. It should also be pointed out that since the data-sets contained only traffic produced by the attacker, following the model's logic and approach a successful result would be the majority of the users to be identified as collaborators, of course this is inline with the precondition that the number of timesteps is high enough to avoid multiple packets on the same interval, as well as inaccurately increasing the comparable frequency.

4.2.1 Low-rate CAIDA DDoS

Here we will test 3 of the 13 CAIDA files, that can be consider as low-rate DDoS since the attacker's packet rate per second is in the range of 1,000. The files that will be tested are:

- *ddos2007_135936*
- *ddos2007_140436*
- *ddos2007_140839*

As already mentioned each file has 300 second of DDoS attack traffic with the peak packet rate being about 1600 packets per second. On figure 4.1 an example traffic graph

taken from wireshark I/O graph can be seen, which proves that the data-sets contain low-rate DDoS traffic.

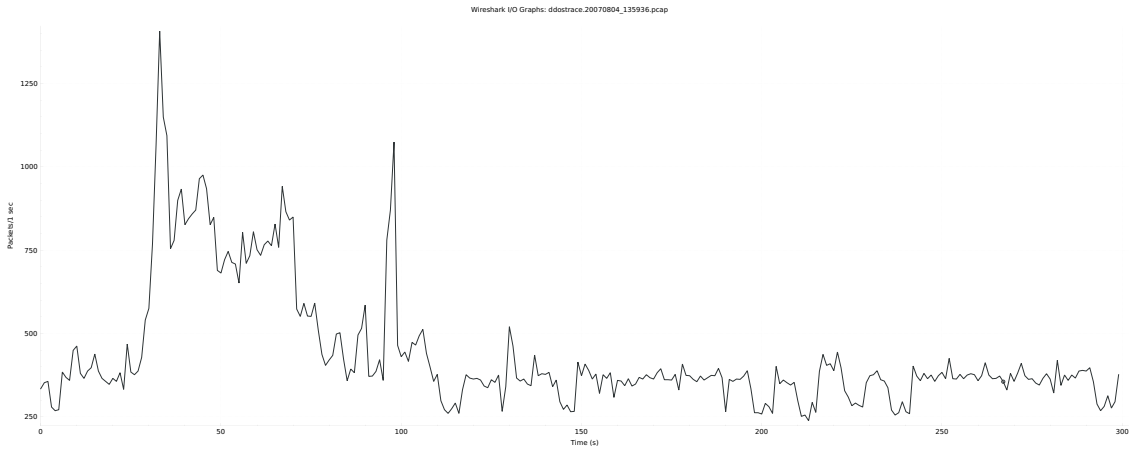

Figure 4.1: ddos2007_135936 Wireshark I/O graph

Below the tables with the result of each data-set can be seen. The results were taken from the implementation described at section 3 but in-order to make sure that no false-observation are made, they were also checked with the original implementation which confirmed their correctness by resulting in similar results (no entirly the same due to the reason described at section 3.2.2). : At table 4.2 the results of the data-set 135939 can be seen, were it contain a total of 118049 packets and 87 unique IPs. Table 4.3 shows the results of the data-set 140839 can be seen, were it contain a total of 105861 packets and 24 unique IPs Finally table 4.4 shows the results of the data-set 140436, were it contain a total of 109170 packets and 11 unique IPs

Time-interval / metrics	**Total timesteps**	**Collaborators (users %)**	**Non-Collaborators (users %)**
3 seconds	100	62.07	37.93
2 seconds	150	54.02	45.97
1 seconds	300	33.33	66.66
100 msec	3000	14.94	85.06
10 msec	30000	4.60	95.40
1 msec	299995	1.15	98.85

Table 4.2: ddos2007_135939 Results - 87 users

Time-interval / metrics	Total timesteps	Collaborators (users %)	Non-Collaborators (users %)
3 seconds	100	87.50	12.50
2 seconds	150	83.33	16.66
1 seconds	300	70.83	29.16
100 msec	3000	41.66	58.33
10 msec	30000	20.83	79.16
1 msec	299995	4.16	95.83

Table 4.3: ddos2007_140436 Results - 24 users

Time-interval / metrics	Total timesteps	Collaborators (users %)	Non-Collaborators (users %)
3 seconds	100	90.90	9.09
2 seconds	150	63.64	36.36
1 seconds	300	63.64	36.36
100 msec	3000	54.54	45.45
10 msec	30000	36.3	63.64
1 msec	299995	54.54	45.45

Table 4.4: ddos2007_140839 Results - 11 users

Conclusions: From the above tables we can conclude that the accuracy detection model is completely and entirely depended on the selection of the time-interval. This can be seen on all of the tables above as with the time-interval increasing, we observe that the accuracy decreases, even though we have more timesteps and should in theory reached the steady state of the detection, we in fact observe the opposite. Another major conclusion is that with increasing users we also have a lower accuracy since the users are more possible to be independent most of the time, and even if they are found depended for multiple times the algorithm will dictate them as independent.

4.2.2 High-rate CAIDA DDos

Similarly to the previous subsection, we will be testing the 4 different CAIDA files that include a high-rate DDoS attack. Each of the selected files includes traffic with average packet rate at the scale of 100 000 packets per second (as shown in which is more than enough to be considers a high-rate attack. Since the model exhibits a quadratic time complexity, specifically $O(n^2 * m)$ where n corresponds the number of unique users and m the number of timesteps, the complexity indicates that its execution time increases proportionally with the square of the number of the unique users. For this reason, it was decided to follow the approach of the original paper, where instead of using the whole data-set to produce the results, a small sample of the data-set is used with size of 1000 packets. Although this is not optimal, using the whole data-set files to produce the results introduces a lot of problems and could be considered a data science problem, this observation is one of the major disadvantages of implementing and using the current model in real world scenarios and will be taken largely into consideration on the evaluation section.

Due to the high amount of traffic in the files that we will be testing, it was not possible to use the same time-intervals that were used in the low-rate testing of the model, simply due to the fact that many user activity will be overlooked as the intervals used before are too large for the amount of traffic per second in this data-set. The author of the paper also came into the same realization and in the original implementation that I received the timestamps were multiplied by a factor of 10 000 in-order to achieve a comparable timestep. This is equivalent of using a time interval in the class of nano-seconds and essentially proves on its own that an optimum time-interval for this model cannot exist.

As already mentioned each file has 1000 packages and the files that we will be testing are:

- *ddos2007_141436* : 9 Unique IPs, results shown at table 4.5
- *ddos2007_142936* : 926 Unique IPs, results shown at table 4.6
- *ddos2007_144936* : 903 Unique IPs, results shown at table 4.7
- *ddos2007_145436* : 916 unique IPs, results shown at table 4.8

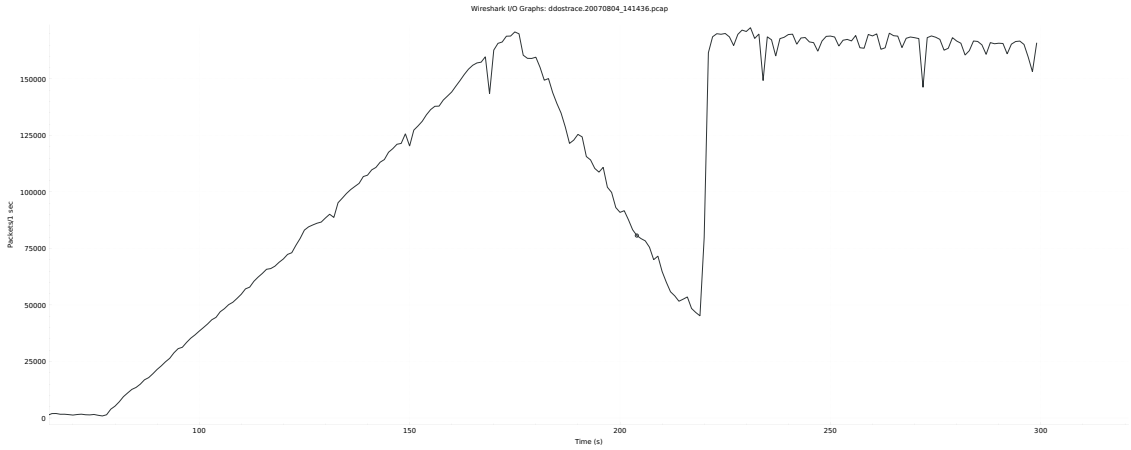

Figure 4.2: ddos2007_141436 Wireshark I/O graph

Time-interval / metrics	**Total timesteps**	**Collaborators (users %)**	**Non-Collaborators (users %)**
10 msec	287	55.55	44.44
1 msec	2858	44.44	55.55
100 nsec	28568	11.11	88.88
10 nsec	285668	22.22	77.77
1 ns	2856665	0	100

Table 4.5: ddos2007_142936 Results - 9 users

Time-interval / metrics	**Total timesteps**	**Collaborators (users %)**	**Non-Collaborators (users %)**
10 msec	2	100	0
1 msec	8	99.78	0.22
100 nsec	62	94.70	5.30
10 nsec	610	66.74	33.26
1 nsec	6083	0	100

Table 4.6: ddos2007_141436 Results - 926 users

Time-interval / metrics	Total timesteps	Collaborators (users %)	Non-Collaborators (users %)
10 msec	2	100	0
1 msec	7	99.55	0.44
100 nsec	60	94.02	5.98
10 nsec	583	64.23	35.77
1 nsec	5815	0	100

Table 4.7: ddos2007_144936 Results - 903 users

Time-interval / metrics	Total timesteps	Collaborators (users %)	Non-Collaborators (users %)
10 msec	2	100	0
1 msec	7	99.78	0.22
100 nsec	61	95.20	4.80
10 nsec	592	65.28	34.72
1 nsec	5907	0	100

Table 4.8: ddos2007_145436 Results - 916 users

Conclusions: The conclusion of the high-rate data-set is the same to the conclusion reached on the low rate data-set analysis. This can be seen from the number of timesteps that were used on each case, where for similar number of timesteps we have a similar detection rate. Since in this case we have almost 10 times the amount of users compared to the low rate testing, the accuracy is basically 0 when we have a large number of timesteps.

4.3 Observations

The major observation of this analysis is that a constant time-interval which can be used on different data-sets to accurately identify collaborative users does not exist. But most importantly the detection model's detection rate seem to decline exponentially with the increase of timesteps and number of users, the reason for this could be easily explained by considering the structure timestep matrix and the algorithm's logic respectively.

An observation about the correlation between the timestep number and accuracy, that could be logically realized by understanding the detection's logic is that with an increase in the number of timesteps, meaning a smaller time-interval, the activity of a user is more finely represented in the timestep matrix. This results in a harder identification of

collaborators since a small time difference in a user's activity could result in having a completely different active timestep. In contrast, by having a small number of timesteps (small time-interval) user is more possible to be active at the same timestep as other users since each timestep represents a larger duration for users to be active in.

Regarding the correlation between the number of users and accuracy, by understanding the implementation logic we can observe that with a smaller number of users the cases that the user is found depended has more "weight" in the overall identification of the user. To understand this, imagine the simple scenario of having 100 non-collaborative users and 10 collaborative users, the algorithm provided in the original paper clearly states that, in-order for a user to be considered a collaborator (depended), the user must be found as depended more times than independent. In this simple scenario even though the user is in collaboration with 10 users, the result would be to be identified as independent as the number found in collaboration is 10 times smaller than the number found being non-collaborative. This posses a major flaw in the logic of the model, that is believe to be overlook by the author.

Chapter 5

Model evaluation

This section will include the final evaluation of the model, by considering all of the aspects of the model and the observations made both from the analysis but also from the implementation of the model. After excessively, researching and fully understanding the logic of the collaborative misbehavior model, the final conclusion is that the model can not be used in-order to detect any collaborative activity at its current form and further research as well as practical experimenting needs to take place for this model to come as close on DDoS detection as effective proven detection methods.

5.1 Uniqueness of this method

This model in comparison with other DDoS detection models, has the uniqueness of being relying only in the utilization of the only observable information available to us, namely the network requests made by users, in order to detect any anomalous activities, with out the need of packet inspection or any other information.

Considering this is the innovative part of this detection model, the question that should be asked is if getting more data that are easily available to a DDoS detection model inside a network infrastructure is considered to be inefficient. For example, getting the packet size or other information that is easily available could benefit in differentiating a normal user by an abnormal one and even help the detection model accuracy by lowering the number of users that need to be compared.

5.2 Complexity

With the detection model having a complexity of $O(n^2 * m)$ where n and m is denoted as the number of users and m the number of time-steps, it is deemed very hard to be considered as a detection model that can be used in today's infrastructures containing 1000 of users as the computation would require capable hardware and with fast execution time being a necessity for quick decision making. This makes the model in its current state a computationally inefficient solution, as it was seen in the high-rate CAIDA data-set testing, were the total execution time of a minute of network traffic required a day (although the data-set contain a very high packet-per-second rate which is definitely abnormal in normal operations).

A possible solution for the computation intensity of this method, is to make use of the parallelism of the program in-order to achieve a substantial decrease in the execution time of the program, by using common parallel programming techniques like using vector registers, multi-threading architectures and GPU programming like CUDA, which could easily be incorporated in the python implementation.

5.3 Time interval selection

As already mentioned in the abstract of this thesis, the initial goal of this project was to identify a time-interval constant for which maximizes accuracy of the model. The conclusion of the analysis using the CAIDA data-set, revealed the logical realization, that such a value does not exist. The reason being, that the model's underlying algorithms or mathematical techniques may not be able to adequately capture and differentiate the nuanced distinctions among the timestamp values. Consequently, the identification process becomes challenging or even impossible when dealing with extremely small differences in timestamps. In contrast, there is the possibility of selecting a time-interval for which, small differences in user activity may be overshadowed, causing the model to incorrectly classify pairs as similar. This can result in a higher occurrence of false positives, where user activity that is actually dissimilar is erroneously identified as similar due to the model's low time-step number which causes user activity to be more possible to occur at the same time-step.

The criteria which is thought to be the most accurate when selecting a time-interval, is to select an interval as close to the data-set's mean packet rate. The reason being that since the mean packet rate represents the packet rate of most users, by selecting a time-interval based on that value, the time-steps are more possible to be able to match the user activity on a unique timestamp.

5.4 Collaborator Identification

As it was explained extensively throughout this thesis, the collaborator identification is not based sorely on the comparison of the relative and comparable frequency, but rather a more generalized rule is used in the practical implementation where a user to be collaborative must be found more times as a collaborator with other users rather that a non-collaborator. This is a clear assumption of the original author, and resulted in a very inaccurate results in the analysis section, simply because it is highly unlikely for every user having the same exact traffic even if all users are instructed by the same botnet master due to other overlooked variables that occur in general networking operation (like delays and re-transmissions).

More importantly this is a very insufficient criteria of identifying a group of collaborators in the normal operations of network, since most definitely the users in the collaboration group will be found as non-collaborators most of the time in normal network traffic. The criteria will result in establish that the users are collaborators since the times found in collaboration were less than those found as non-collaborators.

Chapter 6

Future Work

In this section we will mention areas on which further research can be done, concerning the collaborative misbehaviour model, even though the outcomes obtained from the analysis did not meet the desired level of satisfaction. First of all, the algorithmic logic model has to be re-evaluated as in its current state cannot be deemed as a valid DDoS detection model. More specifically, the model's collaborative identification that compares the times a user found in collaboration and non-collaboration, is as proven an invalid criteria moving forward, and a more concrete identification needs to be researched and implemented.

Furthermore, a future research could be the creation of weighted relationships between users, where the collaboration with a user that is more possible to be part of a DDoS attack, will have more weight in comparison to the comparison with other users. This could be done additionally to also adding more information in the model, to identify users as abnormal not with just making use of the timestamp of the user's requests.

An aspect that was inadvertently overlooked in both research studies is the analysis of the model's accuracy when applied to a more complex dataset encompassing traffic originating from both DDoS attackers and normal users. This examination is crucial to demonstrate the practical applicability of the model in real-world scenarios where the ability to differentiate between normal users and DDoS attackers holds significant importance. By evaluating the model's performance in such diverse and realistic datasets, researchers can ascertain its effectiveness and reliability in distinguishing malicious traffic from legitimate user behavior. This evaluation would not only validate the model's viability in practical environments but also provide valuable insights into its robustness and potential limitations when confronted with complex and heterogeneous network traffic. Incorporating an assessment of accuracy under these circumstances will enhance the comprehensiveness and practical relevance of the research findings.

Moving forward, future work for this research will focus on enhancing the implementation efficiency by leveraging parallel programming techniques and other strategies such as vector registers. This is particularly crucial due to the observed complexity of $O(n^2 * m)$, which indicates a quadratic relationship between the input size and the computational time required. By employing parallel programming paradigms, such as utilizing multiple cores or distributed computing resources, the computational tasks can be divided and executed concurrently, leading to significant performance improvements. Additionally, exploiting vector registers can allow for efficient data manipulation and processing by leveraging SIMD (Single Instruction, Multiple Data) operations. By optimizing the implementation through these techniques, the aim is to reduce the overall computational complexity and enhance the efficiency of the research's proposed methodology.

Chapter 7

Conclusion

In conclusion, this research has examined the proposed detection methodology and evaluated its effectiveness in accurately identifying and distinguishing collaborative misbehaviour users in a DDoS attack setting. The findings of this study have shed light on several shortcomings of the current state of the detection methodology, highlighting its limitations and inadequacies in achieving accurate results.

The analysis revealed that the proposed methodology falls short in accurately differentiating between users that are collaborative in a network by having similar traffic with non-collaborative users. The model's performance exhibited significant deficiencies, leading to a notable number of false positives and false negatives. This lack of accuracy raises concerns about the practical usability of the methodology in real-world scenarios, where the precise identification of DDoS attacks is of utmost importance.

Moreover, the research uncovered certain challenges associated with the complexity of network traffic patterns, particularly when confronted with diverse and heterogeneous datasets. The detection methodology is proved to struggle to adapt and accurately classify instances in scenarios where the traffic patterns exhibited substantial variations or overlapped with legitimate user behavior.

These findings indicate the need for further improvements and refinements in the detection methodology. Future research endeavors should address the identified shortcomings and explore alternative approaches to enhance the accuracy and robustness of DDoS attack detection. Possible avenues for improvement include the incorporation of advanced parellel programming techniques, utilization of more comprehensive and diverse datasets, and the exploration of innovative algorithms to better capture the unique characteristics of DDoS attacks.

In summary, while the proposed detection methodology serves as a starting point for identifying and mitigating DDoS attacks, this research has conclusively demonstrated its current limitations and lack of accuracy. The identified shortcomings underscore the significance of ongoing research and development efforts to advance the state-of-the-art

in DDoS attack detection and ensure its practical viability in real-world scenarios.

Bibliography

[1] Caida ddos 2007 attack. `https://catalog.caida.org/dataset/ddos_attack_2007`.

[2] Ddos attack detection and wavelets.

[3] Reflective ddos. `https://catalog.caida.org/dataset/ddos_attack_2007`.

[4] C. Douligeris and A. Mitrokotsa. Ddos attacks and defense mechanisms: classification and state-of-the-art. *Computer Networks*, 44(5):643–666, 2004.

[5] K. Li, W. Zhou, P. Li, J. Hai, and J. Liu. Distinguishing ddos attacks from flash crowds using probability metrics. In *2009 Third International Conference on Network and System Security*, pages 9–17, 2009.

[6] J. Mirkovic and P. Reiher. A taxonomy of ddos attack and ddos defense mechanisms. *ACM SIGCOMM Computer Communication Review*, 34, 05 2004.

[7] D. Moore, C. Shannon, D. J. Brown, G. M. Voelker, and S. Savage. Inferring internet denial-of-service activity. *ACM Trans. Comput. Syst.*, 24(2):115–139, may 2006.

[8] M. Rehman, S. Raza, and A. Masood. Ddos attacks in the application layer: A review. *Journal of Network and Computer Applications*, 75:423–440, 2016.

[9] A. Shevtekar, K. Anantharam, and N. Ansari. Low rate tcp denial-of-service attack detection at edge routers. *IEEE Communications Letters*, 9(4):363–365, Apr. 2005. Funding Information: Manuscript received August 16, 2004. The associate editor coordinating the review of this letter and approving it for publication was Dr. Chuan-Kun Wu. This work has been supported in part by the New Jersey Commission on Science and Technology via NJWINS. The authors are with the Advanced Networking Laboratory, ECE Dept., NJIT, Newark, NJ (e-mail: ansari@njit.edu). Digital Object Identifier 10.1109/LCOMM.2005.04008. Copyright: Copyright 2018 Elsevier B.V., All rights reserved.

[10] M. Thoma and C. N. Hadjicostis. Detection of collaborative cyber-attacks through correlation and time dependency analysis. In *2016 18th Mediterranean Electrotechnical Conference (MELECON)*, pages 1–6, 2016.

[11] M. Thoma and C. N. Hadjicostis. Detection of collaborative misbehaviour in distributed cyber-attacks. pages 28–41, 2021.

[12] A. Vikram and Mohana. Anomaly detection in network traffic using unsupervised machine learning approach. In *2020 5th International Conference on Communication and Electronics Systems (ICCES)*, pages 476–479, 2020.

.1 Appendix A: Original Implementation flash crowd detection

In the code snippet below, the original implementation has a "PINAKAS_EIDIKOU_BAROUS" matrix were it contained a mixture of variables, including the comparable frequency (column 2), relative frequency (column 3), if the user was part of a flash crowd (column 4 indicated with 1)

```
1
2
3 f3=find(prob_MATRIX_PROBABILITY(:,2)>=mean_packet_rate);
4 length_f3=length(f3);
5 for i=1:length_f3
6     PINAKAS_EIDIKOY_BAROYS(f3(i),4)=1;
7     %O SIMPLIRONETE O SINTELESTIS BAROYS GIA TO
          mean_packet_rate (FLASH CROWD) STIN STILI 4 TOY
          PINAKAS_EIDIKOY_BAROYS
8 end
9
10 %%BASIKI LOGIKI. H TIMI POY EMFANIZETAI STIN STILI 4 (FLASH
     CROWD) KAI STIN STILI 3 (DEPENDENT), O SYNDIASMOS TOYS MAZI
      EXEI MEGALYTERO EIDIKO BAROS APO TIN TIMI POY EMFANIZETAI
     STIN STILI 2 TOY PINAKA EIDIKOY BAROYS (INDEPENDENT). AYTO
     PRAKTIKA SIMAINEI OTI OTI STIS PERIPTOSEIS AYTES H TIMI TIS
      STILIS DEPENDENT MIDENIZETAI OSTE NA ISXYEI H IERARXISI
     FLASH CROWD PIO SIMANTIKI APO DEPENDENT H LOGIKI EINAI OTI
     AYTOS POY EINAI FLASH CROWD SIGOYRA EINAI DEPENTENT.
11
12     f5=find((PINAKAS_EIDIKOY_BAROYS(:,4)==1)&(
          PINAKAS_EIDIKOY_BAROYS(:,3)>PINAKAS_EIDIKOY_BAROYS(:,2)
          ));
13     False_negative_f5=find((PINAKAS_EIDIKOY_BAROYS(:,4)==1)&(
          PINAKAS_EIDIKOY_BAROYS(:,3)>PINAKAS_EIDIKOY_BAROYS(:,2)
          ));
14     f5_clear_flash_crowd=find(PINAKAS_EIDIKOY_BAROYS(:,4)==1)
          ;
15     False_positive=find((PINAKAS_EIDIKOY_BAROYS(:,4)==1)&(
          PINAKAS_EIDIKOY_BAROYS(:,2)>PINAKAS_EIDIKOY_BAROYS(:,3)
          ));
```

.2 Appendix B: Implementation of the model in python

```
import math
import sys
from collections import OrderedDict

def process_file(filename, time_interval):
    seen_ips = set()
    unique_ips = []
    timestamps = []
    with open(filename, 'r') as file:
        for line in file:
            parts = line.strip().split()
            timestamp = float(parts[0])
            source_ip = parts[1]

            if source_ip not in seen_ips:
                unique_ips.append(source_ip)
                seen_ips.add(source_ip)

            timestamps.append((timestamp, source_ip))



    min_timestamp = timestamps[0][0]
    max_timestamp = timestamps[-1][0]
    num_columns = math.ceil((max_timestamp - min_timestamp) /
        time_interval)+1

    matrix = [[0 for _ in range(num_columns)] for _ in range(
       len(unique_ips))]

    sum_matrix = [[0] * num_columns for _ in range(len(
       unique_ips))]

    ip_index = OrderedDict((ip, i ) for i, ip in enumerate(
       unique_ips))


    for timestamp, source_ip  in timestamps:
```

```
35          column = math.floor((timestamp - min_timestamp) /
                time_interval)
36          row = ip_index[source_ip]
37          matrix[row][column] = 1
38          sum_matrix[row][column] += 1
39
40
41      mean_packet_rate = len(timestamps)/(max_timestamp-
            min_timestamp)
42      print("Total packets: ",len(timestamps))
43      return matrix, sum_matrix, unique_ips, mean_packet_rate
44
45
46
47  def compute_probabilities(matrix, unique_ips):
48      num_ips = len(unique_ips)
49      num_columns = len(matrix[0])
50      # counter [i][1] = counts time user was depended
51      # counter [i][2] = counts time user was independed
52      counter = [[0, 0] for _ in range(num_ips)]
53      depended = set()
54      independed = set()
55      for i in range(num_ips):
56          ip1 = unique_ips[i]
57          if ip1 in depended:
58              continue
59          for j in range(i+1, num_ips):
60              ip2 = unique_ips[j]
61
62              count_ip1 = sum(matrix[i])
63              count_ip2 = sum(matrix[j])
64              count_both = sum(1 for column in range(
                    num_columns) if matrix[i][column] >= 1 and
                    matrix[j][column] >= 1)
65
66              prob_ip1 = count_ip1 / num_columns
67              prob_ip2 = count_ip2 / num_columns
68              prob_both = count_both / num_columns
69              if prob_ip1 == prob_both and prob_ip2 ==
                    prob_both:
```

```
                depended.add(ip1)
                depended.add(ip2)
            elif prob_both > (prob_ip1*prob_ip2):
                counter[i][0] +=1
                counter[j][0] +=1
            else:
                counter[i][1] +=1
                counter[j][1] +=1

        if ip1 in depended:
            continue
        elif counter[i][0] >= counter[i][1]:
            depended.add(ip1)
        else:
            independed.add(ip1)

    return depended, independed




print("
    ============================================================"
    )
filename = "input.txt"  # Replace with the actual filename
time_interval = float(sys.argv[1])  # Time interval in
    seconds
timesteps_matrix, sum_packet_matrix, ip_table, packet_rate =
    process_file(filename, time_interval)

depended, independed = compute_probabilities(timesteps_matrix
    , ip_table)
num_ips = len(ip_table)
print("Total unique IPs: ",num_ips)
print("Total timesteps: ", len(timesteps_matrix[0]))
print("Depended: ",(len(depended)/num_ips)*100,"%")
print("Independed: ",(len(independed)/num_ips)*100,"%")
print("
    ============================================================"
```

```
    )
print("Depended; ", depended)
print("Independed; ", independed)
```

Listing 1: "Implementation of the model python code"