Modelling Behavioural Dynamics for Asymmetric Application Layer DDoS Detection

Amit Praseed, *Member, IEEE*, and P. Santhi Thilagam, *Member, IEEE*

***Abstract*— Asymmetric application layer DDoS attacks using computationally intensive HTTP requests are an extremely dangerous class of attacks capable of taking down web servers with relatively few attacking connections. These attacks consume limited network bandwidth and are similar to legitimate traffic, which makes their detection difficult. Existing detection mechanisms for these attacks use indirect representations of actual user behaviour and complex modelling techniques, which leads to a higher false positive rate (FPR) and longer detection time, which makes them unsuitable for real time use. There is a need for simple, efficient and adaptable detection mechanisms for asymmetric DDoS attacks. In this work, an attempt is made to model the actual behavioural dynamics of legitimate users using a simple annotated Probabilistic Timed Automata (PTA) along with a suspicion scoring mechanism for differentiating between legitimate and malicious users. This allows the detection mechanism to be extremely fast and have a low FPR. In addition, the model can incrementally learn from run-time traces, which makes it adaptable and reduces the FPR further. Experiments on public datasets reveal that our proposed approach has a high detection rate and low FPR and adds negligible overhead to the web server, which makes it ideal for real time use.**

***Index Terms*— Application layer, DDoS, asymmetric workload, probabilistic timed automata, incremental learning, suspicion score, anomaly detection.**

I. Introduction

DISTRIBUTED Denial of Service (DDoS) attacks using computationally intensive HTTP requests have emerged as a serious threat to web applications in recent years [1], [2]. These attacks are called asymmetric Application Layer DDoS (AL-DDoS) attacks [3] and are capable of exhausting server resources using considerably fewer attack requests than other application layer DDoS attacks [3], [4].

In addition to their potency, Asymmetric DDoS attacks possess certain features which make them extremely difficult to detect. First, they are executed using legitimate HTTP requests, which makes it impossible to detect these attacks by inspecting individual requests. Second, they exhibit a very low attack bandwidth and thus, cannot be detected by existing volumetric DDoS detection mechanisms. Third, they resemble legitimate user traffic such as flash crowds, which are sudden spikes in legitimate user traffic to a web server due to a noteworthy event or sale [5].

Due to these characteristics, the detection of asymmetric attacks presents a unique and difficult challenge. Any detection mechanism for these attacks must possess certain features that are not necessarily part of other DDoS detection mechanisms. In particular, we present four desirable features for any asymmetric AL-DDoS detection mechanism.

- Lightweight: DDoS detection mechanisms operate alongside the web server, and hence need to be lightweight to avoid becoming a bottleneck that attackers can exploit.
- Judgement: A good detection mechanism must be able to judge whether an incoming request stream is malicious or not with a very low false positive rate (FPR).
- Adaptability: A good detection mechanism must be able to adapt to changing user behaviour. This involves periodic updates to the underlying model to accommodate gradual changes in user behaviour.
- Prioritized Detection: Asymmetric AL-DDoS attacks must be detected faster compared to flooding attacks, because they are capable of bringing down servers with much fewer requests. A good detection mechanism must be able to distinguish between asymmetric attacks and flooding attacks, and must be able to detect the former much faster.

A review of literature reveals that existing detection mechanisms fail to satisfy all four of these criteria. Most of the existing detection mechanisms are computationally expensive which makes their real time use infeasible. They also suffer from drawbacks such as high FPR, and the need for periodic retraining to adapt to changing user behaviour. Another major drawback of existing detection mechanisms is the lack of a prioritized detection mechanism. Most of the existing mechanisms use a common approach for detecting both asymmetric attacks and flooding attacks. This leads to a relatively high detection latency for asymmetric attacks, by which time the server resources may already have been exhausted. Asymmetric attacks need to be recognized as a fundamentally different challenge, and have to be detected faster as compared to flooding attacks.

In this work, we present a novel approach for capturing the behavioural dynamics of users and detecting asymmetric AL-DDoS attacks efficiently. We model user behaviour using an annotated Probabilistic Timed Automata (PTA) and use a scoring mechanism to detect anomalous access to the

Manuscript received January 7, 2020; revised May 26, 2020 and July 18, 2020; accepted August 7, 2020. Date of publication August 19, 2020; date of current version September 10, 2020. This work was supported in part by the Ministry of Electronics and Information Technology (MeitY), Government of India, through the Research and Development Project entitled Development of Tool for Detection of Application Layer DDoS Attacks on Web Applications during the period of 2017–2020. The associate editor coordinating the review of this manuscript and approving it for publication was Prof. Kai Zeng. *(Corresponding author: Amit Praseed.)*

The authors are with the Department of Computer Science and Engineering, National Institute of Technology Karnataka, Surathkal 575025, India (e-mail: amitpraseed@gmail.com; santhisocrates@gmail.com).

Digital Object Identifier 10.1109/TIFS.2020.3017928

TABLE I
SUMMARY OF DETECTION MECHANISMS FOR ASYMMETRIC ATTACKS

Research Work	Mechanism	Judgment	Adaptability	Complexity	Prioritized Detection
Ranjan et al. (2009) [3]	Statistical Distributions			Low	✓
Xie and Yu (2009) [6]	HsMM	✓	✓	High	
Xie and Yu (2009) [7]	HsMM	✓	✓	High	
Wang et al. (2011) [8]	Markov Process, Large Deviation Theory			High	
Giralte et al. (2013) [9]	Statistics, cache graph, path graph		✓	Low	
Xu et al. (2014) [10]	Random Walk Graph			Moderate	
Huang et al. (2014) [11]	HsMM			High	
Emami-Taba et al. (2015) [12]	Game Theory		✓	Moderate	
Meng et al. (2018) [13]	Resource Monitoring		✓	Moderate	✓
Demoulin et al. (2019) [14]	Resource Monitoring		✓	High	✓
Proposed Approach	**Probabilistic Timed Automata (PTA) and Suspicion Scoring**	✓	✓	**Low**	✓

web application. Our proposed approach satisfies all four of the criteria which are necessary for an efficient detection mechanism, namely lightweight, judgement, adaptability and prioritized detection. We also demonstrate that our proposed detection mechanism performs better than existing detection mechanisms.

Our contributions in this work are as follows:

- We propose the use of an annotated Probabilistic Timed Automata (PTA) to capture the behavioural dynamics of legitimate users accessing a web application.
- We propose a mechanism to detect asymmetric AL-DDoS attacks by using cumulative suspicion score assignment based on the annotated PTA.
- We demonstrate that the proposed detection mechanism performs considerably well in detecting asymmetric AL-DDoS attacks, and can be used effectively at real time.

II. RELATED WORK

AL-DDoS detection is an anomaly detection problem, and operates in two stages - learning and detection. During the learning phase, a model representing legitimate user behaviour is constructed using existing server logs. During the detection phase, incoming connections are compared to the learned model, and connections which deviate significantly from the learned model are considered malicious. The efficiency of the system depends on three factors - the features chosen to describe legitimate user behaviour, the model used to represent user behaviour, and how an anomalous access is defined. Detection mechanisms that use statistical features such as request rate, request inter-arrival duration, entropy etc. [15] to model user behaviour are only capable of detecting flooding attacks. Efficient detection mechanisms for asymmetric AL-DDoS attacks need to address two factors - the inclusion of request workload (or a representation of request workload) as one of the features used to model legitimate user behaviour, and a definition of anomalous access that focuses on the request sequence in the incoming connections.

Ranjan *et al.* [3] were one of the first to propose a detection mechanism for asymmetric DDoS attacks by monitoring statistical features in addition to using request workload composition. However, using request composition instead of request sequence does not allow for efficient detection of asymmetric attacks [15]. Subsequent works into asymmetric AL-DDoS detection have used request sequence as one of the primary features, but often fail to incorporate request workload as a parameter. These approaches treat asymmetric AL-DDoS attacks as instances of flooding attacks, and take a significant amount of time to detect them. This could prove dangerous because asymmetric attacks can take down web servers much faster and with fewer requests as compared to flooding attacks.

The works of Xie and Yu [6], [7] were some of the first works to use an HsMM to model user access behaviour. In their works, they used an HsMM to model request sequences and the stochastic variation of page popularity as representations of user behaviour. Their work provided the foundation for the works of Wang *et al.* [8] and Huang *et al.* [11]. HsMM provides an attractive option for asymmetric AL-DDoS detection, but the algorithms associated with detecting request sequence abnormalities are variants of the Viterbi algorithm, and are inherently complex in nature. Deploying such algorithms at run time is not a good option as they could slow down the detection process and degrade the quality of service for legitimate users.

Xu *et al.* [10] modelled user access patterns using a Random Walk Graph and used it to predict the subsequent sequence of requests based on the request history of the user. Any significant deviation from the predicted request sequence is treated as an attack. Giralte *et al.* [9] considered statistical flow characteristics in order to detect application layer DDoS attacks. Emami-Taba *et al.* [12] modelled the scenario of an application layer DDoS attack using Game Theory, and used pay-off tables to decide upon the most suitable course of action for different attacks.

The works of Demoulin *et al.* [14] and Meng *et al.* [13] address a different variety of asymmetric attacks. These attacks use specially crafted (though not illegal) request payloads, such as deeply nested XML sequences, to exhaust server resources. Although sometimes classified as asymmetric attacks, they can be identified and blocked by request inspection and strict schema verification. The context of our work does not seek to address such attacks. Rather, the focus of our work is to address asymmetric attacks where the request sequence, rather than the request payload, forms the basis of attack. The inclusion of these research works is purely for the sake of completeness of the literature review.

A summary of the existing detection mechanisms for asymmetric AL-DDoS attacks are given in Table I. In Table I,

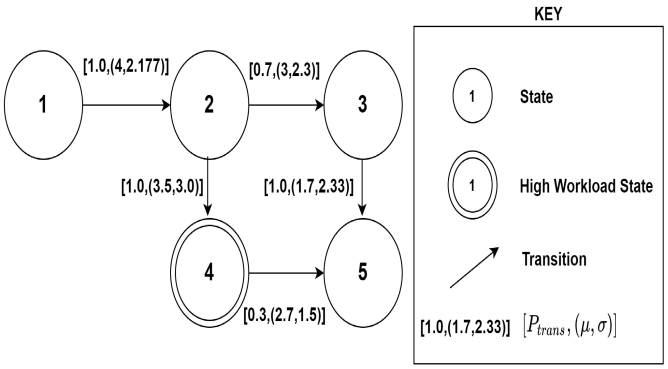


Fig. 1. Diagrammatic Representation of the annotated PTA.

we have marked an approach as having poor judgement if it does not have an FPR value below 1%. The FPR threshold was set to a very low value because unintentional blocking of legitimate users causes a huge loss of revenue to web applications, and hence, it is crucial for DDoS detection mechanisms to have a very low FPR. Table I clearly shows that even though significant work has been done on asymmetric AL-DDoS detection, the available solutions have certain drawbacks. First, majority of the existing detection mechanisms tend to be computationally expensive, which makes them unsuitable for real time use. Second, these approaches are unable to detect asymmetric attacks faster as compared to flooding attacks, which is essential to ensure the safety of the web server. Third, these approaches usually have a high FPR, especially in the presence of flash crowds, and fourth, these approaches require periodic retraining to account for changes in legitimate user behaviour. Thus, there is a need for a more efficient detection mechanism that addresses these drawbacks.

III. Proposed Approach for Asymmetric AL-DDoS Detection

The proposed approach for detecting asymmetric AL-DDoS attacks operates in two phases - learning and detection. In the learning phase, the system uses existing server logs and access traces to model the behavioural dynamics of legitimate users in the form of an annotated Probabilistic Timed Automata (PTA). Once the learning phase terminates, the system can be deployed to detect asymmetric AL-DDoS attacks. During the detection phase, the system intercepts every incoming connection to check if their behaviour deviates from the learned model. Any significant and prolonged deviation from the learned model is considered to be malicious behaviour, and the connection is blocked. Periodically during the detection phase, the system also uses legitimate run time traces to update the learned model. This is done so that the need for periodic retraining does not arise. Figure 2 illustrates the working of the proposed system.

A. Learning Phase

1) Model Description: Probabilistic Timed Automata (PTA) are an extension of Finite State Machines (FSMs) which are capable of modelling probabilistic user behaviour [16]–[18]. Like an FSM, a PTA is a collection of states and transitions. In addition, every transition in a PTA is associated with a weight which denotes the probability of that transition. Every state in a PTA has a time value associated with it, which represents the average amount of time for which the system remains in that state before making a transition.

In this work we model the access patterns in a web application using an annotated Probabilistic Timed Automaton. The URLs or web pages in the web application are represented by states in our model, and the requests between pages are represented by transitions. Every transition has a weight associated with it, which represents the probability of users making that transition. Every transition also has an associated "think time" which represents the amount of time users spend in the previous state before making that transition. We also associate a measure of workload with every state in the model which represents the amount of computation that the server requires for executing that particular request.

Daniele Beauquier introduced a general definition of a Probabilistic Timed Automaton [19] with multiple clocks and actions that can be performed in a state. When modelling a web application, the complexity of the model can be reduced by considering only a single clock maintained by the web server. Also, the actions performed in a state are deterministic, so there is no need to hold on the assumption of multiple actions per state. In our work, we use a simplified model of a PTA with a few annotations.

The annotated Probabilistic Timed Automaton used in our work is a tuple $A = (S, S_{init}, C, W, E, F, \rho, \Phi)$ such that

- S is the set of states in our model, where each state represents a base URL in the web application.
- S_{init} denotes the initial state(s) of the automaton, which is usually the login page of the web application. However, in architectures where there is no concept of separate user logins, every state of the automaton can be considered to be an initial state.
- C is the clock which keeps track of transition timings in our model.
- $W : S \rightarrow \mathbb{N}$ is a priority function which assigns a priority to every state in the model. The priority value is a representation of the computation performed by the web application in that state.
- $E \subset S \times S$ represents the set of transitions in our model. Each edge is a tuple $(s, s\prime)$ such that $s, s\prime \in S$ represent the source and destination states of the transition.
- $F \subseteq S$ is the final state of the automaton. Intuitively, the logout page is the final state of the automaton. However, there are an abundance of scenarios where users simply abandon sessions instead of logging out. To address this situation we consider every state of the automaton as a final state.
- $\rho : E \rightarrow [0, 1]$ is the transition function which maps every transition $(s, s\prime)$ in the model to a probability, which represents the probability of a transition from state s to $s\prime$ in the web application.
- $\Phi : E \rightarrow guard(C)$ is a function mapping every transition $(s, s\prime)$ in the model to a timing constraint within $guard(C)$. $guard(C)$ specifies a set of probability distributions, and Φ maps every transition from s to $s\prime$ to

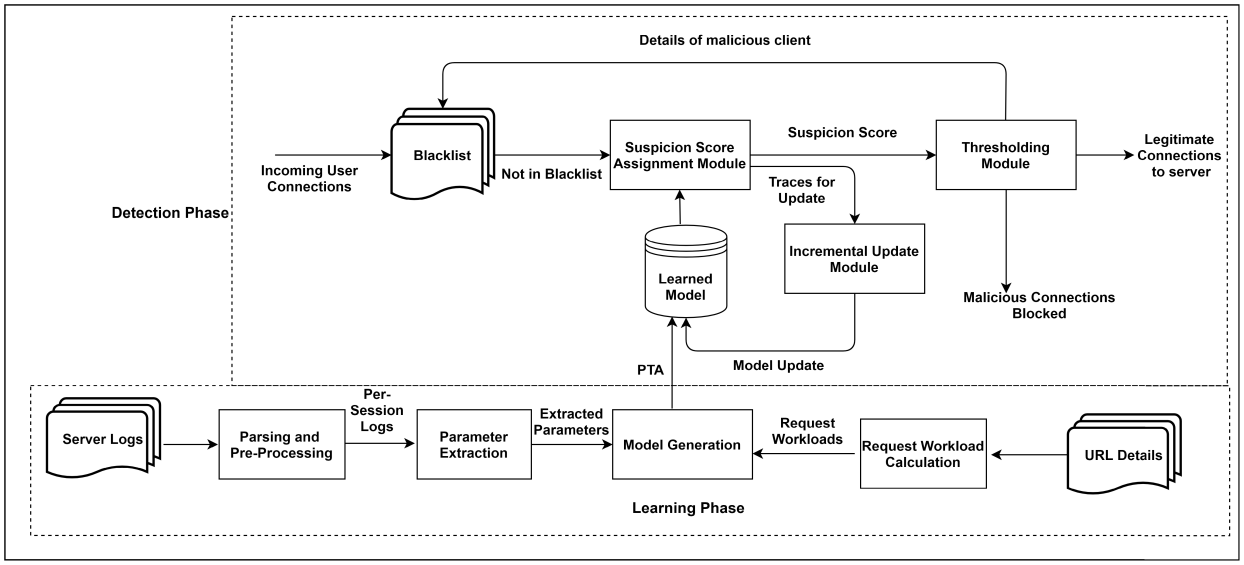


Fig. 2. Block Diagram of the Proposed Approach.

a probability distribution of think times for that transition. A probability distribution is represented by its mean and standard deviation as a tuple (μ, σ).

In our work, the PTA is associated with a scoring mechanism, which assigns a suspicion score for users accessing the web application. A request sequence is said to be accepted by the PTA if it generates a suspicion score below a particular threshold value.

A diagrammatic representation of the annotated PTA used in our approach is given in Figure 1. The states or URLs are given in circles, and the double circles represent high workload states. Every transition has two features associated with it - the transition probability given by ρ, and the associated think time distribution given by Φ. This is specified as a nested tuple $[P_{trans}, (\mu, \sigma)]$ where P_{trans} represents the probability of the transition, and μ and σ represent the mean and standard deviation of the think time distribution for that transition.

2) Suspicion Scoring for Anomaly Detection: When users login to the web application for the first time, they are assigned a Suspicion Score value of zero indicating the highest level of trust. As time progresses, their scores keep changing based on how they access the web application.

The three main features that are used in the suspicion score assignment are

- Transition Probability: Transition probability is an indication of how normal a transition is. A user performing a transition which has a high transition probability is therefore assigned a low suspicion score and vice versa.
- Think Time: Attackers typically try to send requests sequentially without any delay in order to take down systems faster. Thus, if there is a limited time delay between incoming requests as compared to the think time values in the learned model, the access pattern is highly suspicious.
- Workload of the State: Asymmetric DDoS attacks are executed by sending requests to high workload states in the web application. Since such attacks can exhaust server resources much faster than other AL-DDoS attacks, it is necessary to detect these attacks faster. In order to do this, we incorporate a quantity representing request workload into the suspicion score assignment.

A legitimate transition is usually associated with a high transition probability (P_{trans}) and a legitimate think time value (P_{think}) when compared to the think time value for that transition. Since these two events are independent, the probability that an access is legitimate is given by

$$P(LegitimateAccess) = P_{trans} * P_{think} \tag{1}$$

Using Equation 1, the probability that a user access is anomalous can be given by

$$P(AnomalousAccess) = 1 - P(LegitimateAccess) \tag{2}$$

$$= 1 - P_{trans} * P_{think} \tag{3}$$

The probability of the transition can be directly obtained from the PTA. The PTA maintains the mean value of the think time for transitions and their associated standard deviations. If the incoming transition occurs with a think time of t, then a probability that this think time is legitimate can be derived from Chebyshev's Theorem.

$$P_{think} = \left(\frac{\sigma}{\mu - t}\right)^2 \tag{4}$$

To ensure prioritized detection, we introduce a factor that represents the workload of the state that the user is attempting to access, which is represented by W. Thus, the increment in suspicion score for every request that the user makes is given by

$$SS_{increment} = W * (1 - P_{trans} * P_{think}) \tag{5}$$

3) Algorithm and Working of the Learning Phase: During the learning phase, the system uses server logs which describe the workload associated with individual requests, and access logs which describe user behavioural dynamics in order to build the annotated PTA. Algorithm 1 describes the process of learning in the form of pseudo code.

Algorithm 1 Learning Phase

Input: I_+, a training set of per-session request sequences
Output: Annotated Probabilistic Timed Automata (PTA)

Initialize $pta = (S, s_{init}, C, W, E, F, \rho) = \phi$
while LEARNING **do**
  $I = ReadLine(I_+)$
  $curr \leftarrow ExtractState(I)$
  if $curr$ is initial state **then**
    $s_{init} \leftarrow curr$
  end if
  if $curr$ is final state **then**
    $F \leftarrow F \cup curr$
  end if
  if $curr \notin S$ **then**
    $S \leftarrow S \cup curr$
    $w \leftarrow ExtractPriority(curr)$
    $W \leftarrow W \cup (curr, w)$
  end if
  if $(prev, curr) \notin E$ **then**
    $think\_time \leftarrow ExtractThinkTime(curr, prev)$
    $t \leftarrow CreateTransition(prev, curr, think\_time)$
    $E \leftarrow E \cup t$
  else
    $t \leftarrow ExtractTransition(prev, curr)$
    $UpdateTransition(\rho, t)$
  end if
  $prev \leftarrow curr$
end while
$\theta \leftarrow GetThreshold(pta)$

User access logs contain interleaved requests from various users. The preprocessing phase groups user access logs based on client IP address to get access logs for individual users. These per-user access logs are further split into shorter lists called sessions. Each session contains the access logs for a particular user in a certain time interval. The duration of a session varies widely, and it is common practice to consider a session to have ended after a certain period of inactivity, called a session timeout. In our work, we have used a session timeout value of 10 minutes as is common in Apache servers. These per-session access logs (I_+) are passed as input to the Learning Phase for constructing the model. The learning phase starts with an uninitialized PTA, and the Parameter Extraction module extracts the required parameters to build the PTA. When a previously unseen URL is encountered in the access logs, a new state is created in set S to represent this state. A state is associated with a workload value, which is extracted from the server logs with the help of the $ExtractPriority()$ function [3], [20].

A transition is defined as a jump from $prev\_state$ to $curr\_state$. If such a transition is previously unknown, the $CreateTransition()$ function is called which computes two parameters. The first parameter is the transition probability which is maintained in ρ. In any per-session access logs, if there is an entry for a request to a URL i followed immediately by a request to URL j, there is a transition from state i to state j in the model of the web application. The transition probability is computed by counting the number of transitions between any two states i and j. Once all the per-session logs are exhausted, the probability of the transition can be given by

$$t_{ij} = \frac{n_{ij}}{\sum_{k \in S} n_{ik}} \tag{6}$$

where n_{ik} represents the number of times there is a transition from state i to state k in the per-session access logs. The second parameter that needs to be extracted relates to the think time exhibited by users for a particular transition. For a transition instance, the associated think time can be extracted by simply subtracting the timestamps corresponding to $prev\_state$ and $curr\_state$. This gives us a distribution of think time values for every transition, which can be concisely represented using the mean (μ) and standard deviation (σ) of the distribution. In case a transition describing the access pattern already exists, the parameters are updated by the $UpdateTransition()$ function.

This process is repeated until the set I_+ is exhausted. Once the learning is done, it is necessary to extract a threshold value for the suspicion score values, which is done by the $GetThreshold()$ function, which uses the suspicion scoring mechanism described in Section III-A.2. The $GetThreshold()$ function generates suspicion score values for all legitimate client sessions in I_+. Using this suspicion score distribution of legitimate users, a suitable threshold can be estimated above which the all client accesses are marked as malicious by the detection algorithm. Different thresholding techniques can be used depending upon the suspicion score distribution [21]–[23]. Once the parameters and threshold are computed, they are stored in the database for future use. This effectively concludes the learning phase.

B. Detection Phase

During the detection phase, the system processes all incoming connections to the web application like a reverse proxy. All incoming connections are subjected to cumulative suspicion score assignment as described in Section III-A.2. At any point of time, if a client exhibits a suspicion score value greater than the threshold θ, that client is marked as malicious and blocked.

1) Incremental Update: Concept Drift is a fundamental challenge in modelling user behaviour in web applications [24]. User access patterns and popularity of web pages change over time and this gradual shift in user behaviour renders models obsolete after a point of time. When this happens, the model has to be retrained with new access patterns, which requires a long time and considerable computation even though only a small part of the model may have changed significantly. We propose an incremental learning approach using testing data, where the system uses user traces obtained during testing to periodically update the model of the web application. This can be done at regular intervals, or when the load on the system is low.

In order to update the model, we fix a suspicion score threshold, Θ_{update}, such that $\Theta_{update} < \Theta$, below which all

users are assumed to be legitimate. These traces are used to build a model of the web application at run time. Let us assume that for a transition $(s, s\prime)$, we obtain a transition probability $p_{ij}\prime$ during runtime using testing traces. Also, let p_{ij} be the transition probability of the same transition in the current model. In that case, after the next update,

$$p_{ij}(new) = \alpha * p_{ij}(old) + (1 - \alpha) * p_{ij}\prime \tag{7}$$

where α is an anchor value which denotes how much the system should retain the initial values learned. A larger anchor value means the system behaviour changes slowly in response to observed changes in user behaviour, while a smaller anchor value allows rapid changes to the model. In addition, the think time values associated with a transition can be updated by using simple batch update arithmetic [25].

2) Algorithm and Working of the Detection Phase: During the detection phase, the system intercepts every request coming to the web application. Whenever new users enter the system, they are assigned a suspicion score of zero. Henceforth, every access made by users to the web application causes their suspicion score to change cumulatively based on whether the access is perceived to be legitimate or malicious. If the suspicion score associated with a user crosses a predetermined threshold value θ, the user is considered to be malicious and blocked. The incorporation of a suspicion scoring mechanism allows a slight deviation from the learned model to occur, which leads to a lower FPR. The calculation of suspicion score involves three parameters that are easily identified from the PTA - transition probability, think time probability and state priority. The calculation of suspicion score is performed as described in Section III-A.2.

At suitable points during the detection phase, the system also uses traces from legitimate users to update the user model built during the learning phase. This is done in order to avoid the need to periodically retrain the model to account for concept drift. The traces selected for the update are from users who have suspicion scores below a threshold θ_{update}, $\theta_{update} \ll \theta$. This allows only legitimate traces to update the model, thereby maintaining the integrity of the model. This is done as described in Section III-B.1. Algorithm 2 presents a pseudo code for the detection phase.

Algorithm 2 Detection Phase

Input: Request R, PTA, Threshold θ
Output: Decision (Blocked or Allowed)

$updatetraces \leftarrow \phi$
while True **do**
  Input new request R
  $curr \leftarrow ExtractState(R)$
  $w \leftarrow ExtractPriority(s)$
  $think\_time \leftarrow ExtractThinkTime(prev, curr)$
  if $(prev, curr) \in E$ **then**
    $t \leftarrow ExtractTransition(prev, curr)$
    $p_{trans} \leftarrow \rho(t)$
    $p_{think} = GetThinkTimeProbability(t, think\_time)$
  else
    $p_{trans} = p_{think} = 0.0$
  end if
  $ss = (1 - p_{trans} * p_{think}).w$
  $SS = SS + ss$
  if $SS \geq \theta$ **then**
    The connection may be filtered
    $updatetraces \leftarrow updatetraces - R$
  else
    The connection can be forwarded to the server
  end if
  if $SS \leq \theta_{update}$ **then**
    $updatetraces \leftarrow updatetraces \cup R$
  end if
  $prev \leftarrow curr$
  if Update Condition is met **then**
    $Update(pta, updatetraces)$
  end if
end while

IV. Results and Discussion

A prototype of the proposed system was developed using the Go programming language. The performance of the system was evaluated using two popular datasets which describe user access behaviour - SDSC-HTTP and CLARKNET-HTTP. SDSC-HTTP contains a day's worth of all HTTP requests to the SDSC WWW server and contains around 30,000 lines of access logs. CLARKNET-HTTP contains two week's worth of all HTTP requests to the ClarkNet WWW server and contains around 13,00,000 line of access logs.

A. Attack Generation

None of the existing DDoS attack generation tools are capable of generating Asymmetric DDoS attacks. The lack of attack logs is a common issue in the domain of DDoS attack detection, and a number of research works have addressed this issue by generating attack logs from legitimate access logs [26]–[28]. The attack traces in our experiments were generated synthetically by the process outlined in these research works, along with the method suggested by Ranjan et al [3]. A workload profile of the web application is created by monitoring the response times for different requests. Response time of a request can be considered as an approximation of the request workload when network delay is assumed to be constant. Once the request workload profile of the web application is generated, asymmetric attacks are generated by choosing the high workload requests. We generated a total of five different attacks. The first attack (A1) is a random flood, which randomly sends requests to the web server. The second attack (A2) is the Single URL flood, which is the most common flooding attack. Instead of randomly selecting from the available URLs, it sends repeated requests to a single URL. Attacks A3 and A4 are variations of attacks A1 and A2 using only high workload requests. Attack A3 is a random asymmetric attack and A4 is a single URL asymmetric attack. Attack A5 is referred to as a Botnet based attack due to its similarity to DDoS traffic

TABLE II
TYPES OF ATTACKS GENERATED FOR TESTING

Attack Code	Attack Type	Request Workload	No. of URLs (Single/Multiple)	Request Generation Sequence
A1	Random Flood	Any	Multiple	Random
A2	Single URL Flood	Any	Single	Fixed
A3	Random Asymmetric	High Workload	Multiple	Random
A4	Single URL Asymmetric	High Workload	Single	Fixed
A5	Bot Attack	Any	Multiple	Fixed

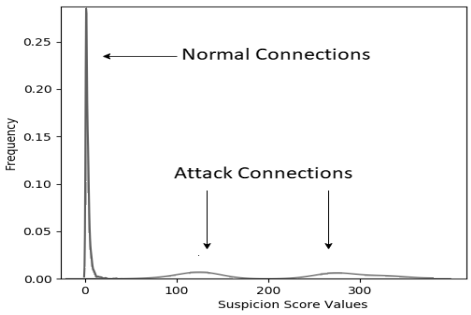


Fig. 3. Suspicion Score Distribution.

TABLE III
PERFORMANCE OVERVIEW OF ATTACK DETECTION MODULE

Parameter	SDSC-HTTP	CLARKNET-HTTP
Detection Rate (DR)	99.8%	99.7%
False Positive Rate (FPR)	0	0.008%
False Negative Rate (FNR)	0.0015%	0.003%
Precision	100%	99.93%
F1 Measure	0.9989	0.9981

executed using a botnet. This attack initially creates an attack script by randomly selecting a set of URLs. Once the script is generated, the attacking connections repeatedly send requests to all the URLs in the script in the same order. The details of the attacks generated are given in Table II.

B. Attack Detection

a) Validity of the Proposed Approach: The proposed approach assigns suspicion scores to users accessing the web application based on their behavioural dynamics, and the amount of deviation from the learned model. Figure 3 depicts a histogram of suspicion score values for both legitimate and attack connections. It can be observed that the suspicion scores for legitimate users tend to cluster at lower suspicion score values, while the scores for attackers are more evenly spread out with higher suspicion score values in general. This clearly shows that the proposed approach can be used effectively for detecting asymmetric AL-DDoS attacks.

b) Performance of the Detection Mechanism: Table III gives the overall performance of our approach on different datasets. It can be observed that the proposed approach gives a very high value of precision and recall along with a very low false positive rate. The performance of our detection mechanism on different classes of attacks individually is presented in Table IV. While most of the attacks could be detected with an accuracy of 100%, there were a few cases where the detection mechanism was unable to detect the incidence of an attack. This is largely due to the incomplete set of traces used for modelling legitimate user behaviour. The completeness of the model and the efficiency of the detection mechanism are dependent on the size and quality of the input logs. Unlike existing detection mechanisms, our proposed approach assigns suspicion scores to connections based only on its individual attributes and not on global statistics. This makes our approach robust in differentiating between flash crowds and AL-DDoS attacks.

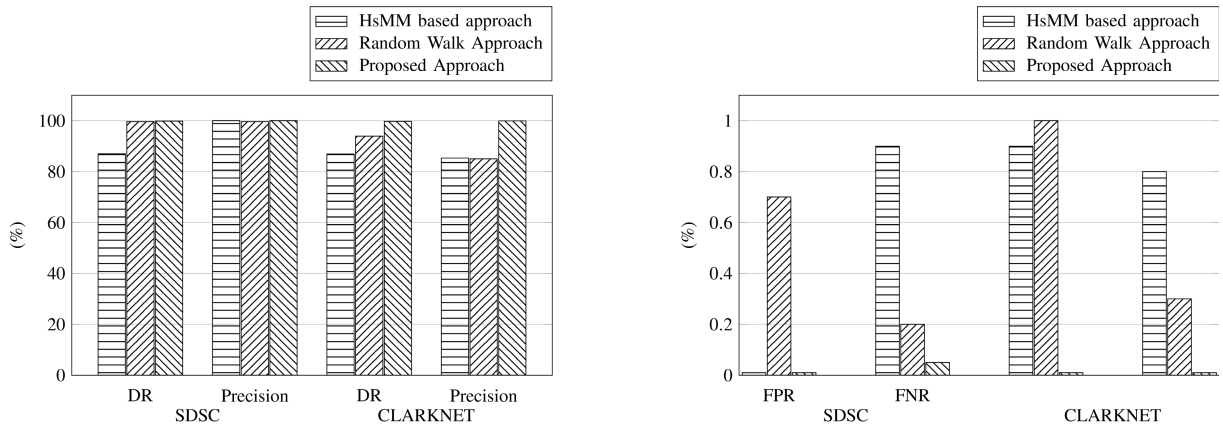
TABLE IV
ATTACK-WISE PERFORMANCE OF ATTACK DETECTION MODULE

Attack Type	Detection Rate (%)	
	SDSC	CLARKNET
Random URL Flood	100	100
Single URL Flood	98	91
Random Asymmetric Attack	100	100
Single URL Asymmetric	98.17	100
Bot Attack	100	100

Figure 4 presents a comparison of our detection mechanism with two prominent research works in the area. We have chosen to compare our work with the HsMM based detection mechanism of Xie and Yu [6] and the Random Walk Model proposed by Xu *et al.* [10]. It can be seen that the proposed work outperforms existing research works in terms of detection rate, precision, FPR and FNR. This clearly indicates that the proposed approach can detect AL-DDoS attacks much more efficiently than existing research works.

c) Complexity: Algorithm 2 describes the detection algorithm used in the proposed approach. It can be observed that to compute the suspicion score for a sequence of N requests, our approach requires only a single pass over the sequence and hence the complexity of the proposed approach is $O(N)$. Existing approaches which use an HsMM have a computational complexity of $O(N^2)$. The use of a modification of the forward algorithm, called the M-Algorithm, can reduce this complexity to $O(MN), M < N$, but still the algorithm is computationally expensive for real time use. Similarly, the Random Walk Model predicts the subsequent states based on the historical request sequence, and requires a complexity of $O(ND)$, where D is the number of states in the model. Thus, our proposed approach clearly outperforms existing detection mechanisms for asymmetric AL-DDoS attacks.

This is further illustrated in Figure 5, which compares the execution time for our proposed approach with existing approaches in a semi log plot. It can be observed that as the sequence length increases, our approach clearly performs better, which further demonstrates that our system is much more suitable for real time use. The actual difference in execution time is considerable - for a sequence length of 50,



(a) Detection Rate and Precision

(b) FPR and FNR

Fig. 4. Comparison with Existing Approaches.

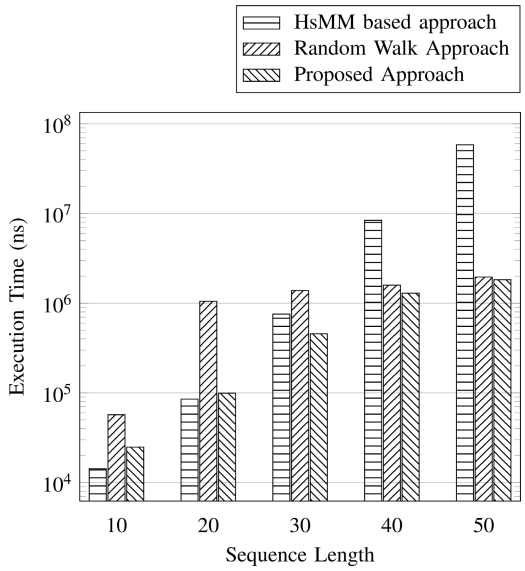


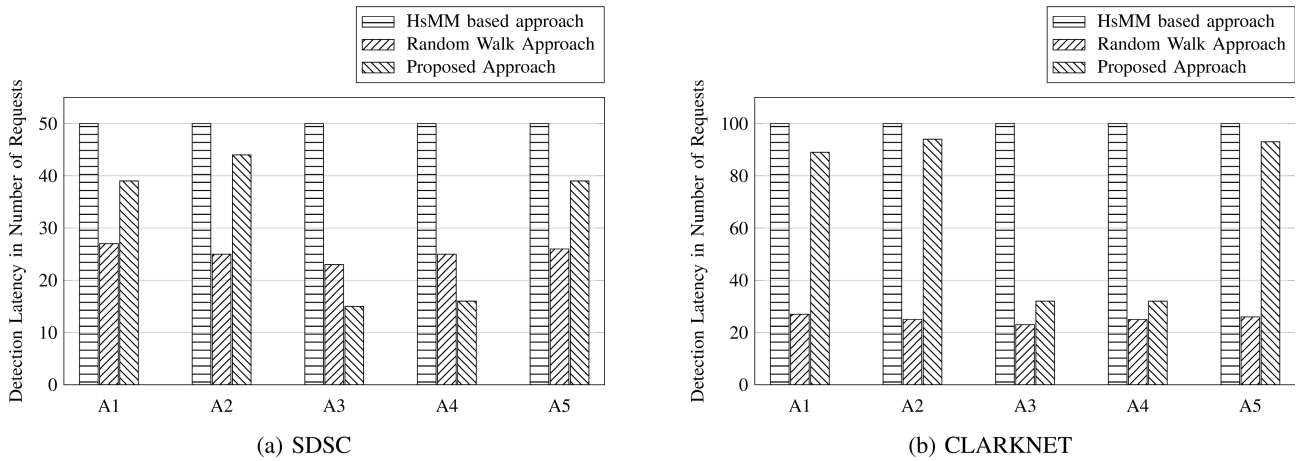
Fig. 5. Execution Time.

an HsMM based approach takes around 60 ms while our proposed approach takes under 2 ms for execution.

d) Prioritized Detection: Figure 6 gives a representation of the time taken for detecting the different classes of attacks compared to the existing approaches. Approaches based on HsMM have a fixed decision length, which denotes the number of requests that need to be inspected in order to make a decision. This means that there is no concept of prioritized detection. Figure 6 clearly shows that our proposed approach detects AL-DDoS attacks faster compared to existing HsMM based approaches. However, the Random Walk approach exhibits a lower detection latency, but at the expense of a higher FPR. Since the Random Walk approach reports an anomaly at the first instance of deviation, it has a lower detection latency, but suffers from a higher FPR. On the other hand, our approach only reports a connection as malicious after repeated infractions, and has a slightly longer detection latency but operates with a significantly lower FPR. It can also be seen that asymmetric attacks (attacks A3 and A4) are detected much faster than HTTP flooding attacks. This is particularly due to the inclusion of the request workload parameter in the calculation of suspicion score. This reduced detection latency is important in the case of asymmetric attacks, because they can take down a system with just a few requests.

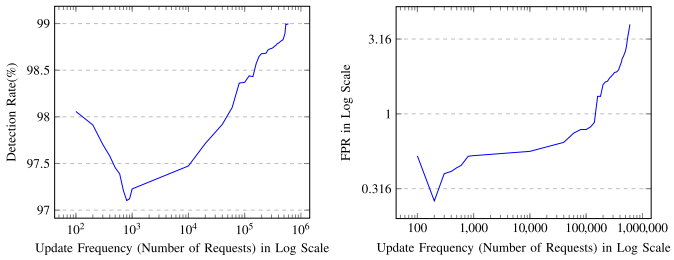
e) Adaptability: The ability to learn new user behaviour and update the model at run time is crucial for asymmetric AL-DDoS detection mechanisms. In our approach, the effectiveness of the update is primarily governed by two factors - the frequency of update and the value of the update threshold. In order to test the performance of the incremental update capacity of our model, we use a modified dataset for simulating concept drift. We separated our dataset into two parts - training and testing. A model of user behaviour was constructed using the training data set. Then, we selected states which were absent in the training data set and generated attack logs using the same approach as described in Section IV-A. We then combined the testing logs with the attack logs to create a new dataset, which we refer to as the $update - test$ dataset. We then set the update threshold to be half of the attack threshold. We monitored the performance of our detection mechanism both with and without update. We observed that even without any update our detection mechanism has a very high detection rate. This is because any unknown state is treated to be an attack in the absence of threshold and the suspicion score is increased as such. However, this means that any legitimate access with previously unknown states is also treated as anomalous. This is indicated by a comparatively large FPR in the absence of any incremental update.

We then tested the performance of our system with periodic updates. It was observed that the update interval plays a crucial role in the performance of the system. If the update interval is set to a small value (say, every 1000 requests), the false positive rate drops considerably, but is also accompanied by a decrease in the detection rate. This is due to the fact that certain attack traces will also be identified as legitimate in the short interval, and be treated as legitimate values. As the update interval increases, the false positive rate remains low, and the detection rate continues to increase. The impact of update frequency on detection rate and false positive rate is shown in Figures 7a and 7b respectively.



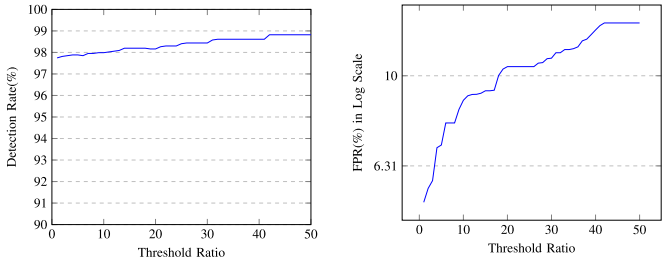
(a) SDSC (b) CLARKNET

Fig. 6. Detection Latency.



(a) Detection Rate (b) False Positive Rate

Fig. 7. Effect of Update Frequency.



(a) Detection Rate (b) False Positive Rate

Fig. 8. Effect of Update Threshold.

In order to study the impact of update threshold on the system performance, we set update threshold to be a fraction of the attack threshold, i.e. $\theta_{update} = \frac{\theta}{k}$, where k is called the Threshold Ratio. Figures 8a and 8b show the impact of the value of k on the detection rate and FPR respectively. Setting a high update threshold allows some malicious traces to be included in the update which leads to a lower detection rate. However, it also shows a lower FPR due to the impact of the legitimate traces used for updating the model. As the threshold is lowered, fewer traces are used in the update (which indirectly means fewer malicious traces are included), and hence the FPR increases. At the same time, the detection rate also rises. This behaviour is similar to that of the impact of update frequency on the detection rate and FPR. The $update-test$ dataset has a high proportion of request sequences in the testing phase that were previously unseen in the training phase. This accounts for the slightly higher FPR demonstrated by the system in this experiment. However, during normal operation, the percentage of previously unknown sequences would be limited, which allows the system to operate at a very low FPR.

The ideal combination of update frequency and threshold ratio to use depends on the system requirements. For systems where a low FPR is a crucial requirement, setting a lower value of threshold ratio and update frequency seems to be ideal. This, however, leads to more frequent updates, which puts additional load on the system. From our experiments, we observed that an update frequency of 10,000 requests and an update threshold value between 2 and 5 allows for an efficient balance between detection rate, FPR and system load.

V. Conclusion

Asymmetric DDoS using computationally intensive HTTP requests is emerging as a serious threat to web applications. These attacks can be executed using very few legitimate HTTP requests which makes it very difficult to detect these attacks. Existing defense mechanisms suffer from certain drawbacks such as high computational overhead, high false positive rate or the need for periodic retraining, which make it unsuitable for real time use. In this work, we have presented a lightweight detection approach for the detection of asymmetric DDoS attacks using an annotated Probabilistic Timed Automata (PTA) for modelling legitimate user behaviour, accompanied by a suspicion scoring mechanism for identifying malicious behaviour. This allows the proposed approach to be lightweight and to have a low false positive rate. The proposed model has an incremental learning capability to adapt to changing user behaviour, which eliminates the need for periodic retraining and makes the model adaptable. We tested our approach using public datasets, which demonstrates that our approach works efficiently to identify asymmetric DDoS attacks much faster, and also has a very low false positive rate. The low complexity of our detection mechanism makes it ideal for real time use.

References

[1] ShapeSecurity. (2016). *Insurer Defeats Application Layer DDoS*. [Online]. Available: https://www.shapesecurity.com/documents/shape_insurance_case_study.pdf

[2] Arbor Networks. (2019). *Netscout Arbor's 13th Annual Worldwide Infrastructure Security Report*. [Online]. Available: https://pages.arbornetworks.com/rs/082-KNA-087/images/13th_Worldwide_Infrastructure_Security_Report.pdf
[3] S. Ranjan, R. Swaminathan, M. Uysal, A. Nucci, and E. Knightly, "DDoS-shield: DDoS-resilient scheduling to counter application layer attacks," *IEEE/ACM Trans. Netw.*, vol. 17, no. 1, pp. 26–39, Feb. 2009.
[4] A. Praseed and P. S. Thilagam, "Multiplexed asymmetric attacks: Next-generation DDoS on HTTP/2 servers," *IEEE Trans. Inf. Forensics Security*, vol. 15, pp. 1790–1800, 2020.
[5] S. Behal, K. Kumar, and M. Sachdeva, "Characterizing DDoS attacks and flash events: Review, research gaps and future directions," *Comput. Sci. Rev.*, vol. 25, pp. 101–114, Aug. 2017.
[6] Y. Xie and S.-Z. Yu, "A large-scale hidden semi-Markov model for anomaly detection on user browsing behaviors," *IEEE/ACM Trans. Netw.*, vol. 17, no. 1, pp. 54–65, Feb. 2009.
[7] Y. Xie and S.-Z. Yu, "Monitoring the application-layer DDoS attacks for popular websites," *IEEE/ACM Trans. Netw.*, vol. 17, no. 1, pp. 15–25, Feb. 2009.
[8] J. Wang, X. Yang, and K. Long, "Web DDoS detection schemes based on measuring user's access behavior with large deviation," in *Proc. IEEE Global Telecommun. Conf. (GLOBECOM)*, Dec. 2011, pp. 1–5.
[9] L. C. Giralte, C. Conde, I. M. de Diego, and E. Cabello, "Detecting denial of service by modelling Web-server behaviour," *Comput. Electr. Eng.*, vol. 39, no. 7, pp. 2252–2262, Oct. 2013.
[10] C. Xu, G. Zhao, G. Xie, and S. Yu, "Detection on application layer DDoS using random walk model," in *Proc. IEEE Int. Conf. Commun. (ICC)*, Jun. 2014, pp. 707–712.
[11] C. Huang, J. Wang, G. Wu, and J. Chen, "Mining Web user behaviors to detect application layer DDoS attacks," *J. Softw.*, vol. 9, no. 4, pp. 985–990, Apr. 2014.
[12] M. Emami-Taba, M. Amoui, and L. Tahvildari, "Strategy-aware mitigation using Markov games for dynamic application-layer attacks," in *Proc. IEEE 16th Int. Symp. High Assurance Syst. Eng.*, Jan. 2015, pp. 134–141.
[13] W. Meng *et al.*, "Rampart: Protecting Web applications from CPU-exhaustion denial-of-service attacks," in *Proc. 27th USENIX Secur. Symp. (USENIX Security)*, 2018, pp. 393–410.
[14] H. M. Demoulin, I. Pedisich, N. Vasilakis, V. Liu, B. T. Loo, and L. T. X. Phan, "Detecting asymmetric application-layer denial-of-service attacks in-flight with finelame," in *Proc. USENIX Annu. Tech. Conf. (USENIX ATC)*, 2019, pp. 693–708.
[15] A. Praseed and P. S. Thilagam, "DDoS attacks at the application layer: Challenges and research perspectives for safeguarding Web applications," *IEEE Commun. Surveys Tuts.*, vol. 21, no. 1, pp. 661–685, 1st Quart., 2019.
[16] F. Abbors, T. Ahmad, D. Truscan, and I. Porres, "Model-based performance testing of Web services using probabilistic timed automata," in *Proc. WEBIST*, 2013, pp. 99–104.
[17] H. Gao, H. Miao, S. Chen, and J. Mei, "Quantitative verification of navigation model for reliable Web applications," in *Proc. 1st ACIS/JNU Int. Conf. Comput., Netw., Syst. Ind. Eng.*, May 2011, pp. 204–209.
[18] C. Ghezzi, M. Pezzè, M. Sama, and G. Tamburelli, "Mining behavior models from user-intensive Web applications," in *Proc. 36th Int. Conf. Softw. Eng. (ICSE)*, 2014, pp. 277–287.
[19] D. Beauquier, "On probabilistic timed automata," *Theor. Comput. Sci.*, vol. 292, no. 1, pp. 65–84, Jan. 2003.
[20] A. K. Mishra, J. L. Hellerstein, W. Cirne, and C. R. Das, "Towards characterizing cloud backend workloads: Insights from Google compute clusters," *ACM SIGMETRICS Perform. Eval. Rev.*, vol. 37, no. 4, pp. 34–41, Mar. 2010.
[21] C. Wang, K. Viswanathan, L. Choudur, V. Talwar, W. Satterfield, and K. Schwan, "Statistical techniques for online anomaly detection in data centers," in *Proc. 12th IFIP/IEEE Int. Symp. Integr. Netw. Manage. (IM) Workshops*, May 2011, pp. 385–392.
[22] A. Siffer, P.-A. Fouque, A. Termier, and C. Largouet, "Anomaly detection in streams with extreme value theory," in *Proc. 23rd ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining*, Aug. 2017, pp. 1067–1075.
[23] L. F. Schneider, A. Krajina, and T. Krivobokova, "Threshold selection in univariate extreme value analysis," 2019, *arXiv:1903.02517*. [Online]. Available: http://arxiv.org/abs/1903.02517
[24] D. Stevanovic and N. Vlajic, "Next generation application-layer DDoS defences: Applying the concepts of outlier detection in data streams with concept drift," in *Proc. 13th Int. Conf. Mach. Learn. Appl.*, Dec. 2014, pp. 456–462.
[25] M. Hancock. (2017). *Batch Updates for Simple Statistics*. [Online]. Available: https://notmatthancock.github.io/2017/03/23/simple-batch-stat-updates.html
[26] C.-H. Lin, J.-C. Liu, and C.-R. Chen, "Access log generator for analyzing malicious website browsing behaviors," in *Proc. 5th Int. Conf. Inf. Assurance Secur.*, vol. 2, 2009, pp. 126–129.
[27] A. Dhanapal and P. Nithyanandam, "An effective mechanism to regenerate HTTP flooding DDoS attack using real time data set," in *Proc. Int. Conf. Intell. Comput., Instrum. Control Technol. (ICICICT)*, Jul. 2017, pp. 570–575.
[28] G. Oikonomou and J. Mirkovic, "Modeling human behavior for defense against flash-crowd attacks," in *Proc. IEEE Int. Conf. Commun.*, Jun. 2009, pp. 1–6.

Amit Praseed (Member, IEEE) received the B.Tech. degree in computer science and engineering from the College of Engineering Trivandrum, India, in 2014, and the M.Tech. degree in computer science and engineering, information security, from the National Institute of Technology (NIT) Karnataka, Surathkal, India, in 2016, where he is currently pursuing the Ph.D. degree in computer science. His research interests include web security and information security.

P. Santhi Thilagam (Member, IEEE) received the B.E. and M.E. degrees in computer science and engineering from the College of Engineering, Guindy, Anna University, Chennai, India, in 1991 and 1999, respectively, and the Ph.D. degree in information technology from the National Institute of Technology Karnataka (NITK), Surathkal, India, in 2008. Since 1996, she has been with the Department of Computer Science and Engineering, NITK Surathkal, where she is currently a Professor. Her current research interests include database security, data management, data analysis, and distributed computing. She is a member of several technical associations, scientific committees, and editorial boards. She was a recipient of the Best Ph.D. Thesis Award in computer science and engineering category of the Board of IT Education Standards in 2009 and the M. S. Ramanujan Lecture Presenter Award of the Institution of Engineers India (IEI-India) in 2015.