Hindawi
Wireless Communications and Mobile Computing
Volume 2019, Article ID 8469410, 14 pages
https://doi.org/10.1155/2019/8469410

*Research Article*

# ForChaos: Real Time Application DDoS Detection Using Forecasting and Chaos Theory in Smart Home IoT Network

**Andria Procopiou**,[1] **Nikos Komninos**,[1] **and Christos Douligeris**[2]

[1]*Centre for Software Reliability, Department of Computer Science, City, University of London, UK*
[2]*Department of Informatics, University of Piraeus, Greece*

Correspondence should be addressed to Andria Procopiou; andria.procopiou.1@city.ac.uk

Received 19 September 2018; Revised 1 January 2019; Accepted 10 January 2019; Published 3 February 2019

Academic Editor: Yu Chen



Recently, D/DoS attacks have been launched by zombie IoT devices in smart home networks. They pose a great threat to network systems with Application Layer DDoS attacks being especially hard to detect due to their stealth and seemingly legitimacy. In this paper, we propose ForChaos, a lightweight detection algorithm for IoT devices, which is based on forecasting and chaos theory to identify flooding and DDoS attacks. For every time-series behaviour collected, a forecasting-technique prediction is generated, based on a number of features, and the error between the two values is calculated. In order to assess the error of the forecasting from the actual value, the Lyapunov exponent is used to detect potential malicious behaviour. In NS-3 we evaluate our detection algorithm through a series of experiments in flooding and slow-rate DDoS attacks. The results are presented and discussed in detail and compared with related studies, demonstrating its effectiveness and robustness.

## 1. Introduction

Smart Homes consist of a great number of different devices, all deployed in a single network monitoring the environment, collecting and sharing important data and information with the owners and other smart IoT devices and external services through internal and external networks. The node responsible for this communication is the Energy Services Interface (ESI). It acts as a bidirectional interface where information can be exchanged between the Smart Home and external domains. Furthermore, it protects internal energy resources from security failures and ensures secure internal communication between the devices deployed in the Smart Home. ESI's importance to the Smart Home and in the outside domains makes it an excellent target for cyberattacks.

DDoS attacks can be conducted across all the layers of the TCP/IP model. Application layer DDoS attacks are much harder to be detected efficiently and accurately than their perspective ones in lower layers as they do not violate any protocol rules or make usage of malicious behaviour. The TCP connections are established successfully and normal requests are sent to the target, in contrast to DDoS attacks in lower layer such as the TCP Flooding which sends a burst amount of SYN packets without acknowledging the SYN,ACK packets sent from the server. The Application Layer Flooding instead sends a burst amount of legitimate requests to the server, which the server cannot refuse but to reply. As a result, it becomes unresponsive due to great amount of incoming requests.

On the contrary, Slow-Rate Application Layer DDoS attack exploits a server's ability to wait for connections to be completed in a range of time, if the incoming connection is legitimately slow. As long as the client manages to send a subsequent packet in an attempt to complete the request the server is obliged to keep the connection open. Based on that, the slow-rate attack opens a great number of connections and initiates requests that never complete them. As time is passing by, more and more connections are open and that results in the target becoming once again unresponsive.

Such attacks can be changed in their form of conduction as there is no fixed behaviour. As a result, traditional signature-based Detection Techniques should fail. Anomaly detection is the obvious solution for detecting Application Layer DDoS attacks, both flooding and slow-rate, in

heterogeneous networks such as the Smart Home network. The potential security system must be able to identify large deviations in the traffic behaviour from the normal behaviour that it is expected but also being robust against temporary normal spikes.

One way of detecting such attacks is to have a holistic behaviour of the attack in terms of time. "Time" is what differentiates this malicious behaviour from normal and can classify it as an attack. The anomaly-detection solution must be able to detect changes in behaviour of the network by monitoring its behaviour over a set of time-series.

Due to the nature of the attacks described above, it is evident that the detection algorithm must monitor closely the present short-term traffic and not base its knowledge on long-term previous behaviour of the system. The reason for this is that due to time being such a volatile entity and the traffic being heterogeneous the network traffic generated can differentiate from to its past history but appear legitimate nevertheless.

Therefore, we must predict what the future short-time-interval traffic is based on present or short-time-interval previous behaviour of the system and not just classify the present network traffic monitoring based on long-term previous fixed behaviour of the system. It is important to avoid incorrectly classifying a behaviour as malicious simply because it is not similar to the history of the network. Our detection algorithm makes use of forecasting techniques to make short-term predictions and identify the attacks through the construction of Lyapunov exponents for every time-series interval.

The remainder of this paper is organized as follows: Section 2 presents the related studies. Section 3 describes the proposed detection algorithm with the appropriate explanation of it. Section 4 explains the simulation testbed implemented for conducting the experiments. Section 5 presents and discusses the experimental results. Finally, a conclusion and future directions are provided in Section 6.

## 2. Related Studies

Various forecasting techniques have been proposed and used in the past for the effective detection of DDoS attacks. The most popular are Moving Average (MA), Weighted Moving Average (WMA), Simple Exponential Smoothing (SES), also called Exponential Weighted Moving Average (EWMA), Double Exponential Smoothing (DES), and Triple Exponential Smoothing (TES) also called Holt's Winter Smoothing. MA and WMA make a forecast solely based on previous observations only while the rest of the techniques consider both past observations and past forecasts.

The authors in [6] used the Box-J Cox Jekings ARIMA model to detect DDoS attacks by monitoring the ip addresses and the packets. They use the Lyapunov exponents to measure the error generated. They have constructed their algorithm to make predictions every 5 minutes with a training time of 100 minutes. They report their algorithm to have generated a TP rate of 94.4%, a FP of 0.1%, and FN of 5.6%.

The authors in [1] used simple exponential smoothing and wavelet analysis in monitoring incoming bytes, number of packets, and the proportion between incoming and outgoing packets to detect UDP Flooding DDoS attacks. They choose window sizes 600 and seconds with an overlap of 10%.

Their algorithm after a series of experiments and adjustments (window size of 100 seconds with an overlap of either 50% of 80%) generates no false positives and manages to detect 10 different types of attack scenarios. However, they do not discuss whether their algorithm generates any FNs in the scenarios, which is quite important, especially since in previous experiments their algorithm was not able to detect the attack at its start time but 200 seconds after initiated.

In [3], the authors proposed linear exponential smoothing to detect Flooding and Scanning DDoS attacks using bytes, flows, and packets per minutes. They report a 24-hour training time and a window size of 5 minutes to accurately detect deviations in traffic. The $a$ value is set to 0.05 and the deviation is measured using entropy. Through their experiments, their algorithm can detect the attack, which lasts for 11 minutes and 13 hours after the algorithm has started being trained. However, it misclassified instances both before and after the actual attack. In total, 11 predictions misclassify normal behaviour as attack out of 288. This gives a FP 3.82%.

In [4] the authors used the triple exponential smoothing to detect TCP SYN Flooding DDoS and Slammer Worm by analysing each packets source ip, destination ip, source port, and destination port. They used a window size of 900 seconds. To assess the error they used of the relative entropy. They detect the attacks inserted in real-traffic from the Brazilian National Research and Education Network which consisted of 5 days of traffic. With a 24-hour training time and a 5-minute window size, the algorithm is able to detect a 1.5 hour TCP SYN-Flooding attack but no further information is given on the time the algorithm detected the deviation.

In [2], the authors used a number of forecasting algorithms, including Moving Average, Weighted Moving Average, Exponential Smoothing, and Linear Regression to predict the intensity and size of DDoS attacks in the TCP protocol such as SYN Flooding, DNS Flooding, and ICMP Flooding. They used a window of 60 seconds between predictions to monitor the number of packets. The two-thirds of the attack, from a total of 53 minutes, were used for training and the remaining one-third was used for testing. To asses which forecasting algorithm has the less error rate they made use of the Absolute Prediction Error metric. For the TCP SYN Flooding 563 packets were generated at each second, for DNS Flooding 8 packets, and for ICMP 1 packet per second. Through the three types of attacks, Exponential Smoothing was the best in detecting the intensity of the attack and size of DNS TCP and ICMP Flooding, while Linear Regression was the most successful in detecting the size of the TCP SYN Flooding. Overall, the algorithm reports to generate an error rate of 1%.

Besides entropy and error to assess the error of forecasts another way to measure error is chaos theory and the Lyapunov Exponents. Chaos theory is an area of mathematics that aims to study nonlinear phenomena that are hard or nearly impossible to predict. More specifically, chaos theory studies dynamic complex systems that are sensitive to initial conditions. A small change in the initial conditions can cause

Table 1: List of features.

| Feature Name | Description |
|---|---|
| Requests No | Total number of requests in a time-series interval |
| Packets No | Total number of packets in a time-series interval |
| Data Rate | Average data rate in Megabits in a time-series interval |
| Avg Packet Size | Average packet size in a time-series interval |
| Avg Time Betw Requests | Average time between two requests in a time-series interval |
| Avg Time Betw Response & Request | Average time between the response and the first requests encountered in a time-series interval |
| Avg Time Betw Responses | Average time between two responses in a time-series interval |
| Parallel Requests | Total number of parallel requests in a time-series interval |

crucial changes in the outcome of the dynamic system. This is also known as the butterfly effect. This highlights that even in deterministic systems, which are entirely dependent on their initial conditions without any random elements involved, their future cannot be predicted. This is described as chaotic behaviour or simply chaos.

In [5], the authors have designed a DDoS detection system which uses self-similarity theory to monitor the traffic and local Lyapunov exponents to distinguish between normal behaviour and DDoS attack. In addition, they use the Lyapunov exponents to train neural networks; they achieve a detection rate of 88-94 % with a false positive of 0.45-0.05 %. Similarly in [8], the authors have once again used chaos theory and Lyapunov exponents in a combination with neural networks to detect DDoS attacks, inspired by [5], achieving a better result with a 98.4% detection rate.

The authors in [7] conduct preprocessing on the network traffic by calculating the simple cumulative average of time series values at every time interval. The local Lyapunov exponent is used to detect a DDoS attack from the DARPA Dataset by using packets flow information. Then they make use of a neural network to improve the DDoS detection accuracy. They report a detection rate of 93.75%.

As discussed in the studies mentioned above, there are multiple ways of assessing the error forecasting algorithms produced on each forecast with the most popular being mean square error, entropy, and Lyapunov exponents. However, there is a major difference between the Lyapunov exponents and the mean square error and entropy. By using either square error or entropy measures, a threshold is assumed, while positive Lyapunov exponents indicate chaos. In forecasting, a positive Lyapunov exponent indicates that the distance between the actual value and the forecasting one is high. The network traffic orbit is both chaotic and unstable. This means that the nearby points diverge to any arbitrary separation, so the change of traffic is due to an attack. A negative Lyapunov exponent shows that the error is not chaotic because the difference between the forecast and the actual observation is small.

A nonchaotic error indicates normal behaviour. The network traffic orbits are attracted to a stable fixed point from when they diverge due to new legitimate traffic, entering the system. Hence, the change of traffic is not due an attack. On the contrary, in case of an attack, the network traffic orbits are not attracted to a stable fixed point.

## 3. ForChaos Detection Algorithm

Below we explain the basic mathematical concepts that have been used in the structure of ForChaos Algorithm.

*3.1. Feature Selection.* Most of the studies using forecasting techniques against DDoS attacks mentioned in the previous section usually make use of a single feature for prediction and detect the possible deviations in the number of packets, the number of packets per IP, the packet flags and so on. However, these studies focus on detecting DDoS attacks on lower layers and not on the application layer. The adoption of one single feature on the application layer is likely not to produce satisfactory results since Application DDoS attacks do not violate any protocol rules and do not produce malformed packets. Therefore we have designed a new set of features to detect Application Layer Flooding and slow-rate DDoS attacks. All of the features (Table 1) are heavily dependent on time and form ideal features for forecasting-based algorithms and presented in Table 1.

*(1) Requests Number*. In a flooding scenario the number of requests over a period of time will be much greater compared to normal traffic request rates. In a slow-rate attack scenario the number of requests over two consecutive periods of time will have large difference. Since slow-rate opens connections to send requests after an amount of minutes there will be a pattern of low number of requests, then high number of requests, then low, and so on.

*(2) Packets Number*. In a flooding attack the number of packets on application layer is rapidly increased over a short period of time. In a slow-rate attack the number of packets is lower due to the packets being sent slowly, just before a request can be rejected. Hence, the request stays alive but very few packets are sent.

*(3) Data Rate*. In a flooding attack the data rate on application layer is rapidly increased over a short period of time. In a slow-rate attack the data rate is lower due to the slow data rate of the malicious requests.

*(4) Average Packet Size*. In a flooding attack the average packet size is decreased due to the requests being simply a get request that features no payload. In a slow-rate attack the average size packet is decreased since a great series of connections

sends small incomplete packets over the attack period of time.

*(5) Average Time between Requests*. In a flooding attack the average time between each request is vastly decreased due to an outburst of requests being sent over a short period of time simultaneously. In a slow-rate attack the average time between each request is decreased compared to the same value in a normal scenario. Instead, there are spikes of the number of requests being sent. When the attack is at the stage of opening new requests, the rate of them is increased. When the attack is at the stage of maintaining the connections open, normal subsequent packets are sent; therefore, the rate of the requests is less than the previous stage.

*(6) Average Time between Response and Request*. In a flooding attack the average time between a response and the next request is vastly decreased. In a slow-rate attack the average time between a response and the next request the slow-rate attack malicious requests cause.

*(7) Average Time between Responses*. In a flooding attack, the average time between two consecutive responses is vastly decreased since a burst of requests is being sent to the server; therefore a great number of responses have to be generated. In a slow-rate attack, the same feature is increased since the slow-rate attack's requests are initiated but never completed.

*(8) Parallel Requests*. In a flooding attack the average concurrent requests are decreased since the attackers send multiple requests that are very quickly finished. In a slow-rate attack, many requests are initiated and stay open over a period of time and the average number of parallel open requests is large.

*3.2. Our Approach to Forecasting.* We choose to exclude MA and WMA because we believe that in order for a forecasting model to make effective and accurate forecasts it must have a balance between past forecasting and observations. Our argument is supported by [9] which makes an evaluation study between MA, WMA, and SES. It concludes that SES is the most effective forecasting algorithm as opposed to the other two against DDoS attacks. Between the three ARIMA-based forecasting models there is one basic difference between each other that assists us into making our final decision. SES has no way of considering trend if there is present in observations while DES considers the potential trend in the formula. TES also considers trend in data and seasonality. Although such assumptions and additional elements are useful in other applications, such as making stock market predictions, there is no need to adopt any of these techniques due to the network traffic being unaffected by any trend or seasonality as the authors in [3] claim. The formula of SES is shown in

$$F_{t+1} = aA_t + (1 - a)\, F_t \tag{1}$$

In an observed time series consisting of observations $A_1, A_2, A_3, \ldots$, for an event, the SES formula $F_{t+1}$ is defined as the next forecast at time $t$. $A_t$ is the previous actual observation, $F_t$ is the previous forecast, and $a$ is defined as the smoothing constant. The smoothing constant a can take the values $0 \leqslant a \leqslant 1$. As it can be observed by (1), the choice of $a$ value plays an important role in having a balance between the forecast and the actual observation. By having the $a$ constant value towards 1, more weight is given to the actual observation. If the $a$ constant value is going towards 0, more weight is given to the past observations. Since we have a set of $F$ features at every time interval $t$, a forecast for each of the features is made based on history.

*3.3. Chaos Approach for Analysing Error.* To measure a system's sensitivity to initial conditions, Lyapunov exponents are used. A Lyapunov exponent of a dynamical system is a metric that describes incredibly small trajectories that are close. Lyapunov exponent is calculated using (4).

It is inevitable that every forecasting system will produce a series of errors between their prediction and the observed values. The error at every time interval $t_i$ is defined by

$$e_i = \left|F_i - A_i\right| \tag{2}$$

At every time interval we will have one forecast for each of the features and therefore an equivalent amount of errors. We define the total error from all the features as

$$E_{total} = \frac{1}{n}\sum_{i=1}^{n} e_i \tag{3}$$

$e_i$ is the error value for every of the $n$ features. In ForChaos $n = 8$, as 8 is the total number of features used.

It is important for these errors to be correctly analysed so malicious behaviour can be accurately and quickly identified. When an error occurs between the actual observer value and the perspective forecast it means that either there is an attack or there is just a temporary unexpected value that is still legitimate. We choose the Lyapunov exponent as the mean to analyse the errors encountered and classify them as chaotic and nonchaotic. The local Lyapunov exponent is calculated as shown

$$\lambda_i = \frac{1}{t_i} \ln \left|\frac{\Delta X_i}{\Delta X_0}\right| \tag{4}$$

A positive Lyapunov exponent indicates a chaotic behaviour at time instant $t_i$., which means the forecast prediction $p_i$ has large distance to the normal observed value $x_i$. The network traffic orbit is both chaotic and unstable. This means that the nearby points diverge to any arbitrary separation, so the change of traffic is due to an attack. On the contrary, a negative Lyapunov exponent shows that the error is not chaotic because the difference between the forecast and the actual observation is small. A nonchaotic error indicates normal behaviour since the network traffic orbits are attracted to a stable fixed point from when they diverge due to new legitimate traffic, or bursty legitimate traffic, entering the system. Hence, the rate of traffic change is not due to attack.

$$\lambda_i = \begin{cases} > 0, & \text{DDoS attack.} \\ \leq 0, & \text{Legitimate Traffic.} \end{cases} \tag{5}$$

1: **Input:** Set of 8 Features f, Time Series t, $\alpha$, Window Size $w$ Traffic State $A_{t-1}$
2: **Output:** alert, not-alert
3: **for** $t = 1$ to n **do**
4: **for** $f = 1$ to 8 **do**
5: $F_t = aA_{t-1} + (1-a)F_{t-1}$
6: $e_t = |F_i - A_i|$
7: $e_{total_t} = e_{total_t} + \frac{1}{8}\sum_{i=1}^{n} e_i$
8: $\lambda_t = \frac{1}{t}\ln\left|\frac{\Delta e_{total_t}}{\Delta e_0}\right|$
9: *If* $\lambda_t > 0$
10: *return alert*
11: *else return not − alert*

Algorithm 1: ForChaos detection algorithm.

Table 2: NS-3 protocols used.

| TCP/IP Layer | Protocols |
|---|---|
| **Physical/MAC Layer** | SH: 802.11 |
| **Network Layer** | SH: IPv4 |
| **Transport Layer** | SH: TCP,UDP |
| **Application Layer** | SH: HTTP, CoAP, MQTT, XMPP, AMQP |

*3.4. ForChaos Algorithm Pseudocode.* Based on the previous mathematical concepts described above, we have constructed the ForChaos Algorithm 1 for detecting Application Layer Flooding and Slow-Rate DDoS attacks in the IoT Smart Home Networks. A specific amount of time is defined as the training procedure. During that time the forecasts are not taken into consideration. Instead, they are used to build the detection model and to make correct predictions. A window size N is chosen at the start. ForChaos Algorithm accepts a continuous $t$ time Series, where $t$ belongs to $T$ as an input. Every $t$ is a $x$. of N window size, where $x$'s value ranges from 1 to infinity. Specifically, if the window size is set to 30, then the T range of values is 30, 60, 90, 120, and so on. At every end of the window size, the values for the eight features discussed in Section 2 are calculated. Based on previous observations, a forecast is made for the value for each feature, according to smooth exponential smoothing. This forecast is compared to the actual value for every feature and an error value is generated. The error is the absolute value of subtracting the actual value from the forecast. The total error is calculated as the summation of errors for all features divided by the errors standard deviation. For that Lyapunov exponent error at that specific Time Series $t$ is calculated, based on (3). If it is positive, then we have a DDoS attack and, if it is negative, then the behaviour is legitimate.

## 4. NS3 Simulation Testbed

*4.1. Smart Home IoT Network Simulation.* For the simulation of the Smart Home network, Network Simulator 3 (NS3) was used. Network Simulator 3 (NS3) is an open-source discrete-event simulator developed in C++. A summary of the networks' parameters and protocols used is given in Table 2.

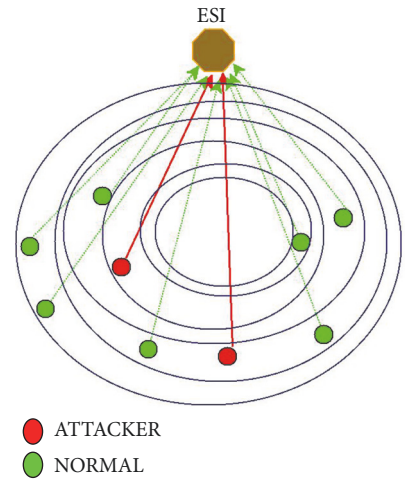


Figure 1: Smart Home architecture in NS3.

A graphical representation of a typical Smart Home is illustrated in Figure 1. Various IoT devices are deployed in the Home Area Network that connect to the gateway, called ESI, to exchange data with the Smart City infrastructures.

Figure 1 forms the representation of the IoT Smart Home simulation in NS-3. 10 nodes were simulated. 9 of them represent IoT devices and Smart Appliances and the final node forms the Smart Home gateway, called ESI. All simulated nodes were static, with no mobility. In the physical layer, all the connections were wireless and the 802.11 protocol was used. The structure was set to ad hoc, using AODV routing protocol. In the network layer IPv4 was used and in the transport layer both UDP and TCP were used. In the application layer a wide variety of application layer protocols were simulated including CoAP, MQTT, XMPP and AMQP, and HTTP.

Our simulation has been constructed to represent realistic Smart Home traffic as much as possible. This has been achieved by using the Smart Home traffic dataset generated by [10]. In [10] they have deployed 30 different Smart Home devices, including smart camers, blood pressure meters,

NEST protect smoke alarm, smart sleep sensors, smart bulb, android/iphone devices, laptops and so on. The maximum amount of packets generated per second was 58.7. Our simulation which consisted of 7 nodes generated an average of 13.7 packets per second.

*4.2. Simulation of Application DDoS Attacks.* Our simulation was designed and implemented based on the specifications found in current smart homes. Smart Homes have currently maximum of 100 Mbps bandwidth. To flood such a line, only 10000 compromised machines will be needed, each capable of sending 1Mbps of upstream. These compromised machines are most likely part of a botnet. The target when attacked will immediately start losing its packets and, with such upstream speed data, it will be unavailable after a minute.

NS-3 simulations are not conducted in real-time. From our experiments, we have estimated that one minute in simulation time is about eight to ten minutes of real-time. Therefore, based on these findings, for the 100 Mbps line to be saturated in simulation it will need only 6-7.5 seconds for a flooding attack. This is evident in our simulation traffic generated. Packets start being dropped after about the 6th second. For a slow-rate attack which is not volumetrically high it needs about 10-12.5 of simulation time for packets to start being dropped. Hence, ForChaos is able to detect the malicious behaviour fast. Furthermore, the attacks are considered inside attacks. Hence, the attackers' capabilities are going to be lighter. The devices compromised are constrained in resources such as memory, processing power, and bandwidth. As a result, an inside attack on the Smart Home will not be able to produce a 1.5 Tbps overhead to the Smart Home network.

Our proposed algorithm's accuracy is evaluated through experiments. Related DDoS detection studies make usage of popular datasets such as the KDD-99, DARPA, or NSL-KDD Datasets. However, we cannot use these datasets as they do not have IoT traffic nor they contain application layer DDoS attacks. In IoT networks the traffic is highly heterogeneous and therefore harder to detect any malicious behaviour. Hence, we had to create our own synthetic traffic using NS-3. A series of traffic containing normal and attack traffic files was generated. Both flooding and slow-rate attacks were simulated. In every scenario, seven nodes from the Smart Home were generating normal traffic and the remaining two were generating malicious traffic.

In order for both normal and attack traffic to be simulated, degree of randomness was added in the network packet generation from the nodes. Randomness was introduced through Poisson Distribution in the following metrics, at the time a packet was created and sent from the client to the server, at the size of the packet generated from both the client and the server, and at the time the server needed to respond. Different speeds were also applied in IoT nodes to service application layer requests. Hence, we can simulate slow connections that are legitimate.

Most of the DDoS attacks, Flooding types in particular, can be classified as constant rate. In constant rate attacks the attackers generate a high steady rate of traffic towards the target [11]. The impact of such an attack is fast, but it can easily be detected due to its obvious intensity. Hence, attackers have moved towards more sophisticated ways of conducting DDoS attacks. One way to evade any security measures installed is to slowly but steadily flood the target in an increasing-rate attack, where the maximum impact of the attack is reached gradually over the attack period. To simulate increased-rate attacks in NS-3, we used open random connections after random time-intervals.

To assess the results on ForChaos algorithm's effectiveness we have calculated the most popular metrics used when assessing a detection algorithm's accuracy. These are Detection Rate(DR), Error Rate (ER), True Positives (TP), False Positives (FP), False Negatives (FN), and Precision. In intrusion detection positives instances are attacks and negative instances are normal. DR measures the algorithm's proportion in correctly classifying incoming instances and ER measures the algorithm's proportion errors in incorrectly classifying incoming instances. DR and ER's sum equals one. TP measures the proportion of positive instances that are correctly identified as such. FP measures the proportion of negative instances to have been misclassified as positive. FN measures the proportion of positive instances that have been misclassified as negative. Lastly, precision measures the proportion of relevant instances among the retrieved instances. In other words, precision measures the rate of true positives divided by all the positives, both correctly and incorrectly classified.

## 5. Results Analysis

Experiments are divided according to the type of the attack and the training time. For every scenario, we have examined out different window sizes and alpha, $a$, constant values from (1). Our main objective was to find the most optimal $a$ and window size with the least training time.

The window size parameter is the time interval at which the detection algorithm generates a new prediction. So, if the window size is 20 seconds, then the algorithm will calculate a new prediction after every 20 seconds. For alpha we tested values from 0.1 to 0.9 with an interval of 0.1 and multiple window sizes between 10 and 60 seconds at an interval of 10 seconds.

After a series of experiments it was observed that the optimal window size is 30 seconds and $a$ is 0.1, as it has generated the least false alarms. Most of the studies using forecasting algorithms take $a$ the value of 0.5, which means that equal weight is given between recent and past observations. However, our integration of chaos with forecasting resulted in a different optimal value of $a$ so that Lyapunov exponents could be calculated correctly.

The training time in all of the attack scenarios was selected to 1000 seconds to accurately detect malicious activities. In total, eight types of attacks have been constructed and the scenarios were divided into flooding and slow-rate attacks.

In the flooding attacks, the parameters differentiated were the number of applications used and the time the applications was initiated. Firstly, 5 applications and then 10 applications were used on each attacker. If the time the applications

Table 3: DDoS flooding experiments results.

| Parameters | Attack Start (sec) | DR (%) | TP (%) | FP (%) | FN (%) | Prec. (%) |
|---|---|---|---|---|---|---|
| **5 apps Const.** | 1000 | 100 | 100 | 0 | 0 | 100 |
| **5 apps Const.** | 2000 | 100 | 100 | 0 | 0 | 100 |
| **5 apps Const.** | 4000 | 100 | 100 | 0 | 0 | 100 |
| **5 apps Incr.** | 1000 | 100 | 100 | 0 | 0 | 100 |
| **5 apps Incr.** | 2000 | 98.6 | 87.50 | 0 | 12.5 | 100 |
| **5 apps Incr.** | 4000 | 100 | 100 | 0 | 0 | 100 |
| **10 apps Const.** | 1000 | 100 | 100 | 0 | 0 | 100 |
| **10 apps Const.** | 2000 | 100 | 100 | 0 | 0 | 100 |
| **10 apps Const.** | 4000 | 100 | 100 | 0 | 0 | 100 |
| **10 apps Incr.** | 1000 | 100 | 100 | 0 | 0 | 100 |
| **10 apps Incr.** | 2000 | 98.6 | 87.50 | 0 | 12.5 | 100 |
| **10 apps Incr.** | 4000 | 100 | 100 | 0 | 0 | 100 |

Table 4: Slow-rate attacks results.

| Parameters | Attack Start (sec) | DR (%) | TP (%) | FP (%) | FN (%) | Prec. (%) |
|---|---|---|---|---|---|---|
| **20 apps(NoSlow)** | 1000 | 100 | 100 | 0 | 0 | 100 |
| **20 apps(NoSlow)** | 2000 | 98.6 | 87.5 | 0 | 0 | 100 |
| **20 apps(NoSlow)** | 4000 | 100 | 100 | 0 | 0 | 100 |
| **20 apps(Slow)** | 1000 | 100 | 100 | 0 | 0 | 100 |
| **20 apps(Slow)** | 2000 | 98.6 | 87.5 | 0 | 12.5 | 100 |
| **20 apps(Slow)** | 4000 | 98.6 | 100 | 1.53 | 0 | 81.82 |
| **40 apps(NoSlow)** | 1000 | 100 | 100 | 0 | 0 | 100 |
| **40 apps(NoSlow)** | 2000 | 97.2 | 100 | 3.13 | 0 | 83.33 |
| **40 apps(NoSlow)** | 4000 | 94.9 | 100 | 5.34 | 0 | 66.67 |
| **40 apps(Slow)** | 1000 | 100 | 100 | 0 | 0 | 100 |
| **40 apps(Slow)** | 2000 | 100 | 100 | 0 | 0 | 100 |
| **40 apps(Slow)** | 4000 | 100 | 100 | 0 | 0 | 100 |

were initiated was set to "constant", then all of the attackers' applications were initiated at the start. If the parameter was set to "increasing," then the attackers' applications were gradually initiated. Through this differentiation, we aimed to test if our detection mechanism was able to identify the attack, even when the peak time of the attack was not visible.

In the slow-rate attacks, the numbers of applications and slow-legitimate connections were integrated in the network traffic as part of its normal behaviour. The number of applications was differentiated from 20 to 40. Through slow legitimate connections, we could evaluate if the algorithm is able to identify malicious or normal activity. In real-traffic, not all connections are fast and completed at once, especially in a Smart Home IoT environment where there are various physical obstacles (e.g., walls). According to [9], there is 26% probability that a connection is slow in a typical home. Furthermore, the study indicates that the more connected devices a home has, the greater the probability there is for the consumers to experience a slow connection. In detail, they report a 47.5% probability of a slow-connection if seven or more nodes are connected to the Home Area Network. Therefore, according to these metrics we have simulated the same percentage as slow-connections being initiated randomly before the actual attack is initiated.

In all scenarios, the attack duration lasted for 200 seconds. For every scenario, the attack is initiated either at the 1000th second, 2000th second or 4000th second. Hence, in total twenty-four experiments were conducted. A summary of the flooding experiments with their parameters and the results are found in Table 3 and for the slow-rate experiments in Table 4.

In our attack scenarios, the target of the attack was the ESI, as it is considered the most important node in a Smart Home.

*5.1. Flash Crowd Scenarios.* Application Layer Flooding DDoS attacks are very similar to Flash Crowd traffic. Flash Crowd denotes the entrance of sudden burst of legitimate traffic in the network that is legitimate. It is hard to distinguish flash crowd events from Application Layer DDoS attacks because both of them generate a high amount of requests in a relatively short amount of time. We wanted to evaluate

Table 5: Flash crowd false positive results.

| Scenario | False Positives (%) | Start of Window Size $t$ Interval |
|---|---|---|
| **FlashCrowd 40 apps 1200sec** | 14.29 | 1110 |
| **FlashCrowd 40 apps 1200sec** | 12.5 | 1380,1860, 1950,1980, 2160 |
| **FlashCrowd 40 apps 4200sec** | 41.12 | 990, 1020, 1110,1170 1260,1380, 1440, 1470, 1500, 1560, 1590, 1620, 1770, 1800, 1830, 1860,1950, 1980, 2040, 2070, 2100,2160, 2190, 2250, 2460, 2490, 2550, 2640, 2730, 2760, 3090, 3210, 3240, 3330, 3390, 3420, 3480, 3690,3750, 3810, 3930, 4020,4110, 4140 |
| **FlashCrowd 45 apps 1200sec** | 85.71 | 990,1020, 1050,1080,1110,1140 |
| **FlashCrowd 45 apps 2200sec** | 82.5 | 990, 1020,1050, 1080, 1110, 1140, 1170, 1200, 1230,1290, 1320, 1380, 1410, 1440,1470,1500, 1560, 1590, 1620, 1650, 1680, 1710, 1740, 1770, 1800, 1830, 1860,1890, 1920,1980, 2040, 2070, 2100 |
| **FlashCrowd 45 apps 2200sec** | 24.30 | 990, 1020, 1080, 1170 1380,1410, 1500, 1650, 1740, 1770, 1830, 1860, 1890, 2100, 2400, 2460, 2550, 2760, 3240, 3420, 3510, 3720, 3840,3900, 4080, 4110 |

ForChaos algorithm through a series of Flash Crowd scenarios. To simulate random FlashCrowd behaviour we used Poisson Distribution. However, the rate of sending packet was increased from the normal behaviour but it is not equal to or exceeding the metrics of the Flooding attack mentioned in Section 5.1. Specifically, we have increased the apps created from 35 to 40 and 45 in the two series of experiments. In the first series of experiments, 40 apps have been used in total in all three experiments which lasted for 1200, 2200 and 4200 seconds prospectively. In the second series of experiments, the same durations have been used but with 45 apps in total in all of them. The results are presented below and summarised in Table 5.

From the results of Table 5, we can observe that ForChaos has difficulty in identifying FlashCrowd events, as it generates a high number of missed alarms. For the experiments that lasted for 1200 and 2200 seconds, ForChaos generates more false alarms in the 45 apps than the 40 apps. This was expected as a larger burst of traffic is likely to confuse the algorithm into misclassifying the behaviour as malicious. Interestingly, in the 4200 seconds scenarios, ForChaos performed better in the 45 apps case than in the 40 apps case. This may have occurred due to ForChaos having calculated more forecasts, therefore becoming more robust in its future predictions.

*5.2. ForChaos Feature Reduction.* As described in Section 3, a total of eight features were used for the detection of Application Layer DDoS attacks. Although the complexity of ForChaos is not high, we have experimented with reducing the features to check whether we can achieve the same detection rate. Instead of randomly reducing the features we have used feature reduction algorithms in our experiments. Nature-inspired algorithms form the most popular feature reduction techniques. Such algorithms followed models from nature, biology, social systems, and life sciences. Some examples include genetic algorithms, swarm intelligence, artificial immune systems, evolutionary algorithms, artificial neural networks, fractal geometry, and chaos theory.

Nature inspired algorithms have an advantage against traditional machine learning algorithms, they focus on optimisation. In detail, nature acts as a method of making something as perfect as possible or choosing the most fitted samples from a population. In practice, this family of algorithms applies these principles in the form of optimisation and finding the best solution to the problem assigned. In anomaly detection, the main objective is to identify the malicious behaviour so these algorithms use their best-fit mechanisms to detect malicious abnormalities. Another beneficial usage of nature/bioinspired algorithms is to optimise the potential features used in attacks detection. In that way, an optimal set of features will be selected for efficient malware detection but also for reducing the complexity and computational burden. Additionally, nature/bioinspired algorithms are highly flexible as they can accept a mixture of variables in terms of type and continuity. This gives us the opportunity to give a variety of different features to the algorithms.

For finding an optimal set of features, we have used the available nature-inspired algorithms provided from Weka toolkit. Specifically, we used evolutionary search, ant and bee search, genetic search, and particle swarm optimisation search algorithms. All of the algorithms used have identified Parallel Requests, Average Data Rate, Average Packet Size, and Packet Number as the optimal features needed. The

Table 6: ForChaos reduced flooding and slow-rate attacks results.

| Parameters | Attack Start (sec) | DR (%) | TP (%) | FP (%) | FN (%) | Prec. |
|---|---|---|---|---|---|---|
| **20 apps NoSlow** | 1000 | 100 | 100 | 0 | 0 | 100 |
| **20 apps NoSlow** | 2000 | 98.6 | 87.5 | 0 | 0 | 100 |
| **20 apps NoSlow** | 4000 | 100 | 0 | 100 | 0 | 100 |
| **20 apps Slow** | 1000 | 100 | 100 | 0 | 0 | 100 |
| **20 apps Slow** | 2000 | 98.6 | 87.5 | 0 | 12.5 | 100 |
| **20 apps Slow** | 4000 | 98.6 | 100 | 1.53 | 0 | 81.82 |
| **40 apps NoSlow** | 1000 | 100 | 100 | 0 | 0 | 100 |
| **40 apps NoSlow** | 2000 | 97.2 | 100 | 3.13 | 0 | 83.33 |
| **40 apps NoSlow** | 4000 | 94.9 | 100 | 5.34 | 0 | 66.67 |
| **40 apps Slow** | 1000 | 100 | 100 | 0 | 0 | 100 |
| **40 apps Slow** | 2000 | 100 | 100 | 0 | 0 | 100 |
| **40 apps Slow** | 4000 | 100 | 100 | 0 | 0 | 100 |
| **20 apps NoSlow** | 1000 | 100 | 100 | 0 | 0 | 100 |
| **20 apps NoSlow** | 2000 | 98.6 | 87.5 | 0 | 0 | 100 |
| **20 apps NoSlow** | 4000 | 100 | 0 | 100 | 0 | 100 |
| **20 apps Slow** | 1000 | 100 | 100 | 0 | 0 | 100 |
| **20 apps Slow** | 2000 | 98.6 | 87.5 | 0 | 12.5 | 100 |
| **20 apps Slow** | 4000 | 98.6 | 100 | 1.53 | 0 | 81.82 |
| **40 apps NoSlow** | 1000 | 100 | 100 | 0 | 0 | 100 |
| **40 apps NoSlow** | 2000 | 97.2 | 100 | 3.13 | 0 | 83.33 |
| **40 apps NoSlow** | 4000 | 90 | 100 | 11.2 | 0 | 33.33 |
| **40 apps Slow** | 1000 | 100 | 100 | 0 | 0 | 100 |
| **40 apps Slow** | 2000 | 100 | 100 | 0 | 0 | 100 |
| **40 apps Slow** | 4000 | 99.3 | 100 | 0.75 | 0 | 87.5 |

results of the reduced ForChaos algorithm across all the scenarios are presented in Table 6.

As it can be seen from Table 6, the reduced ForChaos is able to successfully detect both the Flooding and slow-rate attacks across all different time intervals in all their duration of attack conduction. As it was expected, the reduced ForChaos algorithm was bound to produce a higher level of FP in some scenarios. This occurs in the slow-rate attack scenarios without slow legitimate connections in the 2000 and 4000 seconds. This could occur for the following reason: the reduction of features has made the ForChaos algorithm more sensitive towards changes in the network that are not necessarily malicious.

## 6. Discussion of Results

Throughout the experiments, we proved that our proposed ForChaos algorithm is able to detect malicious activity, with small training time using eight features. However in some experiments, false negatives were identified from our algorithm. Also, certain false alarms were raised under certain experiments.

*Detection Rate*. ForChaos algorithm had the best overall detection rate in the Flooding experiments as opposed to the Slow-Rate experiments. This is due to the nature of the attack itself. Flooding attacks make more "noise" and their behaviour is more obvious than the slow-rate attacks and that is reflected in the features. For example, in a flooding scenario, the request number, average time between each request, and average time between responses and requests are vastly changed in a very short period of time. In Flooding experiments the lowest detection rate is 98.61% while in Slow-Rate experiments the lowest detection rate is 94.93% as shown in Tables 3 and 4.

*False Positives*. Interestingly the ForChaos algorithm generated the most false alarms in the Slow-Rate 40 apps with No-Slow legitimate connections as the overall simulation time was increased. This possibly highlights the weakness of forecasting algorithms in general, of incorrectly predicting values, if too much weight is put into the recent observations. As our $a$ value is set to 0.1, then more weight is put into recent observations, rather than the history. As explained at the beginning of Section 5, through our experiments, the best $a$ value for the Lyapunov exponents correct behaviour was 0.1 though as it generated the least false alarms. For the Slow-Rate 20 apps with No-Slow Legitimate connections no false alarms were raised at all. In general, for the slow-rate attacks the highest number of false positives rate was 5.34% as it is shown in Table 4. The ForChaos algorithm did not generate any false positives after the training time in any of

Table 7: Related studies results.

| Study | Train (sec) | Wnd Size (sec) | Results (DR) | Attack | Features |
|---|---|---|---|---|---|
| [1] | 3200 | 600 | 100 | UDP Flood | Inc. bytes, pckts No, in & out pckts |
| [2] | 3194 | 60 | 99 | SYN Flood, DNS Flood, ICMP Flood | pckts No |
| [3] | 86400 | 300 | 100 | KDD99 | Bytes flows pckts per min. |
| [4] | 86400 | 900 | 100 | SYN Flood | packet src ip, dst ip, src port, dst port |
| [5] | - | - | 94 | KDD99 | pckts No ip adr. |
| [6] | 6000 | 60 | 99.5 | KDD99 | pckts No ip adr |
| [7] | - | - | 93.75 | DARPA | packets |
| [8] | - | - | 98.4 | KDD99 | pckts No ip adr. |
| **For Chaos** | 1000 | 30 | 98.61-100 (Flood) 94.93-100 (Slow-Rt) | App. Flood Slow-Rate | Table 1. |

the Flooding experiments regardless of the attack time. This is due to the nature of the attack itself. Flooding attacks make "noise" and their behaviour is more obvious than the Slow-Rate attacks and that is reflected in the features.

*False Negatives*. In the 5 apps and 10 apps increasing experiments when the attack occurs at the 2000th second, it is not detected in the first $t_i$, where $i$ is the first time series $t$ when the attack is present. However, as the attack progresses the attack is detected in the next $t_i + 1$. Also, in the Slow-Rate 20 apps experiments with or without slow-legitimate connections, the attack could not be initially detected. The reason is that this time only a small number of apps are initiated. Once again as the attack progresses, ForChaos is able to detect that anomaly. The false negative rate was the same across all experiments at 12.5%.

*6.1. Discussion of ForChaos Algorithm Results with Related Studies*. Related studies have been briefly discussed in Section 2 that attempt to perform DDoS detection using forecasting and/or chaos theory and lyapunov exponents with other techniques is summarised in Table 7. We observe that none of them study application layer DDoS attacks but they instead aim to detect their transport layer equivalents.

ForChaos algorithm combines multiple mathematical concepts together to perform detection. To the best of our knowledge, no other study has considered the simple exponential smoothing algorithm and lyapunov exponents to detect DDoS attacks. Therefore, we compare our results with notable studies that make usage of forecasting algorithms or Lyapunov Exponents.

*Detection Rate*. Comparing ForChaos algorithm to the related forecasting algorithms studies we have drawn some interesting results. Regarding training time and window sizes, ForChaos algorithm managed to get a DR of at least 94.3% with just 1000 seconds of Training Time and a 30-second window size as opposed to all the forecasting related studies [1–4, 6] that reported needing a training time varying between 3200 seconds ans 86400 seconds and a window size 60 to 600 seconds to make correct predictions. Therefore, with larger training time and larger window size, the related studies nearly achieve perfect detection rate but ForChaos algorithm follows close enough with 94.93-100%.

In chaos theory, lyapunov exponents are used in combination with neural networks as a replacement to forecasting. The detection rates from the related studies were between 94.05% and 99.5% against DDoS attacks from DARPA and/or KDD-99 dataset. Our various experiments proved our algorithm to have better detection rates across both application layer attack scenarios. Our algorithm has a 94.93-100 % detection rate.

We strongly believe that the main reason for our algorithm's high detection rate with less training time and smaller

window-size is the higher number of features used. The studies presented use between one and four features while we use a total of eight features. This of course increases the complexity of our solution, but we have considered two types of DDoS attacks that are not by any means similar to each other. On the contrary, other studies achieve the same result, making a server unavailable to legitimate requests, by having a vastly differentiated behaviour. All the proposals presented in Section 2 are against mainly flooding attacks and not slow-rate attacks on any network layer. Additionally, the traffic generated from the Smart Home environment is considered heterogeneous both in its normal and in its attacking state.

Therefore, we need to monitor more metrics in order to make accurate classifications about the state of the network, whether it is under attack or not. Application Layer DDoS attacks can have vastly differentiated behaviour, always with respect to time. Therefore, even a large dataset that is going to be used to train the neural network might not be effective when detecting a new type of attack in terms of time. It is essential for the IDS system to be fast and lightweight so it will not constrain the network. Also it is important for the IDS System to be trained with a small dataset fast so it can construct a detection model fast based on the current behaviour of the system and not just its history as the application layer attacks exploit the variable of time and not any protocol rules. Additionally, our algorithm does not need any attack-based dataset to make correct predictions, it just needs a small amount of normal behaviour to be able to detect malicious behaviour as it was illustrated in our results section.

*False Positives*. ForChaos algorithm generates FPs only in slow-rate attacks with the greatest percentage being 5.34%. The other studies which reported FP rates test their algorithm against Flooding types of DDoS only, in which our algorithm does not generate any FPs.

*False Negatives*. ForChaos algorithm generates a FN rate 12.5 % while the only study to report any FN results is [6], with only a 5.6%.

*Complexity*. All of the studies except [1, 3] make use of ARIMA models or the triple exponential smoothing which are heavier in memory and process resources than our own simple exponential smoothing. It is important to take into consideration that most of the devices deployed in a Smart Home IoT environment are resource-constrained, so our detection algorithm is very efficient in comparison to other proposals. Furthermore, neural networks are known to be computationally heavy and require a large dataset to be trained in order to make correct classifications. Artificial Neural Networks need a large dataset for training so they can construct their model consisting of inputs, hidden layers and the outputs, and their connections with each other. In intrusion detection, the most popular dataset used, which is KDD99, consists of 4000000 instances. In the simplest version of an Artificial Neural Network, which is the Multilayer Perceptron, the complexity of classification is roughly $O(n^2)$ with only one hidden layer constructed. However, nearly all constructed models of neural networks have more than one hidden layers so the complexity is increased.

Our algorithm is less complex, as it can be seen from the algorithm pseudocode in Section 2, with a $O(ctf)$, where $t$ is the time series and $c$ is a constant because the number of features is pre-determined, eight. Finally, $f$ denotes the function, in which the forecasting, errors calculation and lyapunox exponenet takes place. It also does not require either a large dataset or a long time to be trained in order to distinguish correctly between legitimate and malicious behaviour. The training time for it is limited to 1000 seconds. Given that it receives a new instance for training at every 30 seconds, it needs roughly 33 instances to start making correct predictions.

To make accurate classification, Artificial Neural Networks need to be trained with both normal and abnormal behaviour in order to distinguish between the two. In addition, our system, as it has been illustrated throughout the diverse experiments, is able to detect various intensities and sizes Application Layer Flooding and Slow-Rate DDoS attacks. On the other hand, Forecasting algorithms do not need both normal and abnormal behaviour to detect malicious behaviour. However, the related studies need much more time than ForChaos. It can be observed from Table 7 that the minimum training time is 3194 seconds and the maximum training time is 86400 seconds (24 hours).

*6.2. Discussion of ForChaos Algorithm Results with Machine Learning Algorithms.* In Intrusion Detection machine learning algorithms form a popular set of techniques, since they can discover patterns in data without any sort of predefined monitored behaviour. Hence, they can perform anomaly detection of unseen attacks without any type of signature. However, no studies have been conducted against Application Layer DDoS attacks in IoT using any Machine Learning Algorithms. Therefore, we have created a dataset of Application Layer DDoS attacks in IoT to evaluate a set of Machine Learning algorithms provided by Weka. The dataset was consisted of the scenarios we have used for the ForChaos algorithm's evaluation. Each raw data scenario was split in 30-second instances and was labelled according to its behaviour (either as malicious or benign). All instances were processed through a feature extraction algorithm to create the dataset. The final dataset file was fed into Weka with the following machine learning algorithms used: Bayesian Networks (BN), Naive Bayesian (NB), Support Vector Machine (SVM), Decision Tree (DT), Random Forest (RF), and Artificial Neural Networks with the Multilayer Perceptron architecture (MLP). Two series of experiments were conducted. In the first series the dataset was randomised, using the "randomisation" filter provided by Weka, before being split into training and test set, with 138 attack instances and 365 normal instances. In the second series of experiments, duplicated instances were removed, through removing "duplicates filter" and then it was randomised through the "randomisation" filter. The dataset consisted of 125 attack instances and 182 normal instances after the duplicates removal. For the training process, in both of the series of experiments, half of the data were used for training the algorithms to construct the models. The results

Table 8: Machine learning algorithms results against application layer DDoS attacks IoT dataset.

| Alg (Dataset) | **DR**(%) | **TP**(%) | **FP**(%) | **FN**(%) | **Prec.**(%) |
|---|---|---|---|---|---|
| **BN(1)** | 95.6 | 86.7 | 1.6 | 13.3 | 94.5 |
| **NB(1)** | 92.4 | 85 | 5.2 | 15 | 83.6 |
| **SVM(1)** | 95.2 | 76.7 | 0 | 13.3 | 100 |
| **MLP(1)** | 98.4 | 96.7 | 1 | 3.2 | 96.7 |
| **DT(1)** | 98.8 | 96.7 | 0.5 | 3.2 | 98.3 |
| **RF(1)** | 98.8 | 96.7 | 0.6 | 3.2 | 98.3 |
| **BN(2)** | 92.8 | 80.7 | 0 | 19.3 | 100 |
| **NB(2)** | 87.6 | 76.7 | 5.4 | 13.3 | 90.2 |
| **SVM(2)** | 87.6 | 68.3 | 0 | 23.7 | 100 |
| **MLP(2)** | 96.7 | 95 | 2.2 | 5 | 96.6 |
| **DT(2)** | 90.2 | 86.7 | 7.5 | 13.3 | 88.1 |
| **RF(2)** | 94.1 | 88.3 | 2.2 | 11.7 | 96.4 |
| **ForChaos** | 94.3 | 87.5 | 5.34 | 12.5 | 81.82 |

are presented in Table 8 and discussed below. The dataset constructed can be made available upon researchers request.

As expected, machine learning algorithms' accuracy was decreased against Application Layer DDoS attacks as opposed to DDoS attacks in lower layers. This is due to the attacks' great similarity to legitimate behaviour and the exploitation of time factor.

*Detection Rate*. MLP has performed best in the second series of experiments while RF performed best in the first series of experiments. Hence, MLP is more effective against unseen behaviour than RF is. Since the duplicates were removed some instances are assessed for the first time. Therefore, MLP is effective in classifying them correctly with a DR 96.7% as opposed to RF's 94.1%. This was expected as neural networks in general are a very accurate and robust classification algorithm. Hence, it was preferred in the related studies. The algorithm that performed worse in terms of detection rate was Naive Bayesian which was expected. NB assumes independence of features which especially in our dataset is not valid.

*True Positives*. For the TP rates, in the first series of experiments RF, DT, and MLP did best as they had the same rate, 96.7%. They were followed by BN, 86.7% and NB, 85%, and SVM 76.7%.

In the second series of experiments, MLP performed best with 95% followed by RF, 88.3%, and DT, 86.7%. MLP is proved to be robust with its TP rate being dropped by only 1.7% while DT and RF had a higher drop rate. RF performed better than DT as expected due to RF's being an "improved" DT version. Also, this difference in the TP rate across the two datasets highlights that the tree or forest being constructed is not as effective as understanding the various types of attack behaviour and their versatility Hence, they fail in correctly classifying them.

For the second series of experiments, the BN and NB methods follow with 80.7% and 76.7% prospectively. From both series of experiments it is evident that probabilistic approaches fail to identify the attack instances. This occurs because probabilistic models need a large dataset to construct accurate and robust probabilities. Also, the removal of duplicates greatly affects their performance as well.

The worst algorithm for TP was SVM with 68.3%. SVM performs worse in both of the experiments. This is due to the small dataset being not enough for the SVM to construct an effective hyperplane.

*False Positives*. In both series of experiments the FP rate was not very high, with the maximum being 7.5%. Specifically, for the first series of experiments SVM did not generate any FPs, followed by DT and RF with only 0.5%. Then MLP generated a 1% of FP rates followed closely from BN with 1.6%. Lastly, NB generated a 5.2% FP rate. In the second series of experiments BN and SVM generated no FP rates, followed by MLP and RF with a 2.2%, NB with 5.4%, and DT with7.5%. In general, all of the algorithms are quite robust against generating FP instances. In fact, they were effectively in understanding when a behaviour is "odd" but still normal. This is evident through the Flash Crowd Scenarios and the slow-rate scenarios where slow legitimate connections are present in the network.

*False Negatives*. In both series of experiments the FN rate was relatively high compared to other studies that used ML algorithms for DDoS attacks in lower layers. Once again, this highlights the nature of the DDoS attacks in the application layer which is an exploitation of time makes them hard to actually detect them.

In the first series of experiments MLP, DT, and RF performed best with 3.3% FN rate. They were followed by BN with 13.3%, NB with 15%, and SVM 23.3%. In the second series of experiments MLP performed best with 5% FN, followed by RF with 11.7%, DT with 13.3%, BN with 19.3%, NB with 13.3%, and SVM with 21.7%.

A high FN rate is a major disadvantage for any IDS system as it means it is unable to identify when an actual attack occurs. The only algorithm that has an "acceptable" FN rate in both series of experiments was MLP. In the first series of experiments RF and DT did well in the first series but their FN

rate was vastly increased in the second series. This indicates that their tree and forest is unable to identify the versatility of the attacks when duplicates are removed. Also, due to the versatility of the attacks and possibly the dataset not being large enough, probabilistic approaches (BN and NB) fail to detect the attacks. Lastly, SVM performs worse as it is unable to construct an optimal hyperplane between the attack and the normal class.

*Comparison with ForChaos*. Through the multiple experiments conducted using ForChaos, the detection rate was varied at 94.93-100 % depending on the attack scenario. At its lowest DR rate, ForChaos manages to pass all of the ML algorithms across the two series of experiments except MLP. Regarding FP and FN rates ForChaos lacks in terms of accuracy as it has a highest 5.34% FP rate while the highest FP rate in ML algorithms was generated by DT, with 7.5% and NB with 5.4%. Regarding FN rates, although ForChaos FN rate (12.5%) is not considered low in general it still is much lower compared to the BN, NB, and SVM in both series of experiments. We also have to point out that the ForChaos algorithm was "trained" using only 33 instances of training while th ML algorithms needed 252 and 154 instances for the two experiments. Also, the ML algorithms to be trained need both attack and normal data while ForChaos only requires normal data to be trained and effectively distinguish between legitimate and malicious traffic. In Table 8, we present the ForChaos minimum values.

## 7. Conclusion

In this paper, we have presented a novel Application Layer DDoS attacks detection algorithm using simple exponential smoothing and chaos theory. Our approach is able to detect both Flooding and Slow-Rate Application Layer DDoS attacks in the Smart Home IoT network. Our proposal is fast and accurate in detecting the attack (10 to 40 seconds after the attack has started), generating a very low number of false positives and it does not require a large dataset to construct the model.

Although, our algorithm proved to have good results there is always room for improvement. Future directions include on attempting to reduce the complexity of our detection method. As already stated, a Smart Home IoT network, or any IoT network for that matter, is low in memory and power so the more lightweight the solution the better. Furthermore, the most false positives generated from our detection engine were under the slow-rate attack scenarios. In particular, a possible future direction is to add more features that have to do more with detecting the slow-rate attack.

Smart Home communicates with many networks and critical infrastructures such as the Smart Grid and VANETS. Each network produces its own heterogeneous traffic so malicious behaviour is going to be different depending on the type, size, and intensity of the attack and what the target is. Hence, cross-network attacks are a likely scenario since so many networks are interconnected together and communicate with each other on a constant and continuous way. Therefore, the Smart Home can be a target of attacks from other networks but it can also participate in large-scale attacks against a Smart City's critical infrastructure, the Smart City itself, or even another country's important assets.

In the future, we aim to protect the communication between the Smart Home and the Smart Grid. It is essential for the Smart Home to be protected from external threats but also from internal threats that aim to abuse its normal functioning and force it to participate in large-scale DDoS attacks that aim to threaten the target critical infrastructure and the Smart City in general.

## Data Availability

The data for constructing the Application Layer DDoS attacks dataset have been generated through simulation tools techniques, specifically NS3. More details on the actual implementation of our simulated environment have been given in Section 4 "NS3 SIMULATION TESTBED." Thus, the data generated are purely synthetic and do not put any individuals' privacy at risk. Relevant parameters regarding the data generation as well as the actual parameters used in simulation have been provided in Section 4. The dataset can be available upon request from the corresponding author.

## Conflicts of Interest

There is no conflict of interest.

## References

[1] P. Shinde and S. Guntupalli, "Early DoS attack detection using smoothened time-series and wavelet analysis," in *Proceedings of the 3rd Internationl Symposium on Information Assurance and Security, IAS 2007*, pp. 215–220, UK, 2007.

[2] C. Fachkha, E. Bou-Harb, and M. Debbabi, "Towards a Forecasting Model for Distributed Denial of Service Activities," in *Proceedings of the 2013 IEEE 12th International Symposium on Network Computing and Applications (NCA)*, pp. 110–117, Cambridge, MA, USA, August 2013.

[3] P. Winter, H. Lampesberger, M. Zeilinger, and E. Hermann, "On detecting abrupt changes in network entropy time series," in *Communication and Multimedia Security lecture Notes in Computer Science*, vol. 7025 of *Lecture Notes in Comput. Sci.*, pp. 194–205, Springer, Heidelberg, Germany, 2011.

[4] A. S. De Moura, "Anomaly detection using Holt-Winters forecast model," in *Proceedings of the IADIS International Conference WWW/Internet 2011, ICWI 2011*, pp. 349–356, Brazil, 2011.

[5] A. Chonka, J. Singh, and W. Zhou, "Chaos theory based detection against network mimicking DDoS attacks," *IEEE Communications Letters*, vol. 13, no. 9, pp. 717–719, 2009.

[6] S. M. T. Nezhad, M. Nazari, and E. A. Gharavol, "A Novel DoS and DDoS Attacks Detection Algorithm Using ARIMA Time Series Model and Chaotic System in Computer Networks," *IEEE Communications Letters*, vol. 20, no. 4, pp. 700–703, 2016.

[7] Y. Chen, X. Ma, and X. Wu, "DDoS detection algorithm based on preprocessing network traffic predicted method and chaos theory," *IEEE Communications Letters*, vol. 17, no. 5, pp. 1052–1054, 2013.

[8] X. Wu and Y. Chen, "Validation of chaos hypothesis in NADA and improved DDoS detection algorithm," *IEEE Communications Letters*, vol. 17, no. 12, pp. 2396–2399, 2013.

[9] Cisco, *Bandwidth Consumption and Broadband Reliability Studying Speed, Performance, and Bandwidth Use in the Connected Home White Paper*, 2012.

[10] A. Sivanathan, D. Sherratt, H. H. Gharakheili et al., "Characterizing and classifying IoT traffic in smart cities and campuses," in *Proceedings of the 2017 IEEE Conference on Computer Communications Workshops, INFOCOM WKSHPS 2017*, pp. 559–564, Atlanta, GA, USA, 2017.

[11] J. Mirkovic and P. Reiher, "A taxonomy of DDoS attack and DDoS defense mechanisms," *Computer Communication Review*, vol. 34, no. 2, pp. 39–53, 2004.