Turkish Journal of Electrical Engineering & Computer Sciences

http://journals.tubitak.gov.tr/elektrik/

Research Article

Turk J Elec Eng & Comp Sci
(2020) 28: 2448 – 2466
© TÜBİTAK
doi:10.3906/elk-1909-112

Real-time anomaly detection and mitigation using streaming telemetry in SDN

Çağdaş KURT[1,*], O. Ayhan ERDEM[2]
[1]Department of Computer Engineering, Institute of Science and Technology, Gazi University, Ankara, Turkey
[2]Department of Computer Engineering, Faculty of Technology, Gazi University, Ankara, Turkey

Received: 20.09.2019 • **Accepted/Published Online:** 20.12.2019 • **Final Version:** 25.09.2020

Abstract: Measurement and monitoring are crucial for various network tasks such as traffic engineering, anomaly detection, and intrusion prevention. The success of critical capabilities such as anomaly detection and prevention depends on whether the utilized network measurement method is able to provide granular, near real-time, low-overhead measurement data or not. In addition to the measurement method, the anomaly detection and mitigation algorithm is also essential for recognizing normal and abnormal traffic patterns in such a huge amount of measured data with high accuracy and low latency. Software-defined networking is an emerging concept to enable programmable and efficient measurement functions for these kinds of challenging requirements. In this paper, we present a new, real-time, model-driven anomaly detection and mitigation platform. Model-driven streaming telemetry and exponential smoothing are the underlying approaches of the platform. A customized collector is proposed to gather streaming telemetry metrics, and Holt's prediction algorithm is improved to handle real-time streaming data and decrease false positives. The developed system is tested on a campus network and the success rate of the system is calculated as 92%.

Key words: Streaming telemetry, anomaly detection, software-defined networks

1. Introduction

Network measurement is a critical matter in many aspects. For any organization that has operations in IT infrastructure and networking, it is a significant job to ensure that the infrastructure is working properly as it is designed for, that its performance is satisfactory, and that nothing happens outside of the network administrator's control. Measurement is also essential to collect data from the network to understand what is happening at a certain time. Thus, network measurement is one of the key topics for real-time intrusion detection and mitigation. Considering the need for provisioning, configuration, and monitoring functions, there are many measurement techniques and anomaly detection methodologies that have been developed in the literature [1–7].

In [5], sFlow and OpenFlow were used to understand traffic patterns in polling and sampling. OF@TEIN [6] was developed for network-wide traffic visibility. sFlow and OpenFlow protocols are used as complementary solutions for collecting information to obtain an overview of a network. FlowTrApp [7] was presented to detect and categorize distributed denial of service (DDoS) attacks by using an efficient conjunction of sFlow and OpenFlow. All three studies [5–7] used statistical metric collection methods to collect flow statistics. Although the studies provided relatively less overhead, if an attack does not reach high traffic rates or it occurs in a short time slot, the sampling-based methods cannot sense the attack. This is the security deficiency of sampling approaches like sFlow and NetFlow in granular attacks.

[*]Correspondence: cagdaskurt@gmail.com

Fully OpenFlow-based studied are presented to overcome the problems of sampling-based methods [8–12]. OpenTM [8] is suggested as a pull-based traffic matrix estimation methodology. It gets statistics and other flow data by using OpenFlow, which induces reduced overhead on hardware resources. PayLess [9] is proposed as an SDN monitoring architecture. By adapting collection frequency between the highest and lowest thresholds, the development aims to provide real-time measurement data with low overhead and high accuracy. In [10], FlowSense was designed for zero measurement cost within its own passive packet capturing approach. Although the proposed architecture brings zero overhead to the network, FlowSense cannot provide real-time monitoring information. It works fine generally when the flow has a short duration.

There are some other studies in the literature that are developed especially for DDoS-based anomaly detection and mitigation [13–20] and that are in the manner of streaming telemetry [21–25]. In [13], an effective traffic measurement solution was proposed to detect network anomalies and malicious attacks. The study focuses on identifying large traffic aggregations to provide high accuracy and low-overhead in measurement. OpenWatch [14] is presented for detection anomaly in software-defined networking (SDN). The objective of the authors was implementing an efficient and interactive application programming interface (API) for anomaly detection platforms. The study focuses on keeping a balance between measurement overhead and anomaly detection accuracy. However, the programmability capabilities are ignored.

The differences between various anomaly detection and mitigation approaches are accuracy, detection speed, granularity, and scalability. Legacy anomaly detection and prevention systems are relatively successful in considering the real-time, programmable, granular, and low-cost measurement requirements of the networks. SDN is an emerging technology in providing a set of advantages to measure, monitor, and manage network flows and hardware [26]. Streaming telemetry is one of the cutting-edge technologies that SDN brings to the networking field as a new measurement concept. It offers structured measurement data to the consumer platforms. The finer granularity and higher frequency of data enable better performance in monitoring and therefore faster action in the case of anomalies. This paper mainly aims at faster anomaly detection and mitigation with higher accuracy in both long-period and granular attacks, higher scalability through many measurement metrics from the devices, more efficiency with less bandwidth usage, and low computational cost.

In this paper, a low-latency, real-time, programmable, granular, and highly accurate anomaly detection and mitigation system is presented. The leverages of the proposed platform are a model-driven streaming telemetry architecture and exponential smoothing prediction approach with distinctive improvements in both areas. An open-source collector is customized to maximize the speed of the transactions in the model-driven telemetry architecture. A new anomaly prediction algorithm empowered by adaptive error constant, service check, and gradual activation functions is presented. The rest of the paper is structured as follows: Section 2 describes all aspects of our algorithms and software modules. In Section 3, evaluation and experimental results are presented. Finally, Section 4 concludes the paper.

2. Proposed work

This study has three main modules as illustrated in Figure 1. The first module is responsible for the modeled data collection. A model-driven streaming telemetry collector is developed by implementing the Google Protocol Buffers (GPB)-Compact encoding mechanism to collect Yet Another Next Generation (YANG)-modeled, GPB-encoded, and Google Remote Procedure Call (gRPC)-transported measurement data from the telemetry device. The second module is responsible for the anomaly detection. Holt's double exponential smoothing forecasting algorithm [27] is enhanced with our developments that significantly affect the results. The third module is for anomaly mitigation after accurate detection. The Network Configuration (NETCONF) protocol is used to deploy predefined prevention policies to the model-driven telemetry device. The rest of the section presents the details and developments of the proposed system.

From the SDN perspective, Figure 2 defines the protocol stack of the planes. Our work relies on data plane telemetry. In this type of telemetry, the quality, quantity, and timeliness streaming abilities of the measurement data are critical for the success of the method. For this reason, model-driven streaming telemetry architecture is applied in the data plane. Because of YANG and NETCONF support, gRPC is used instead of OpenFlow as the southbound transport protocol. NETCONF is used to implement mitigation policies and Python is used to code management plane software and related modules.

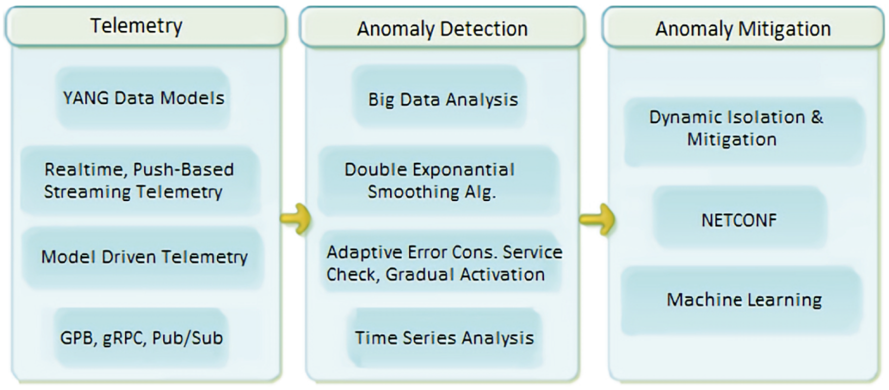


Figure 1. Modules of the proposed work.

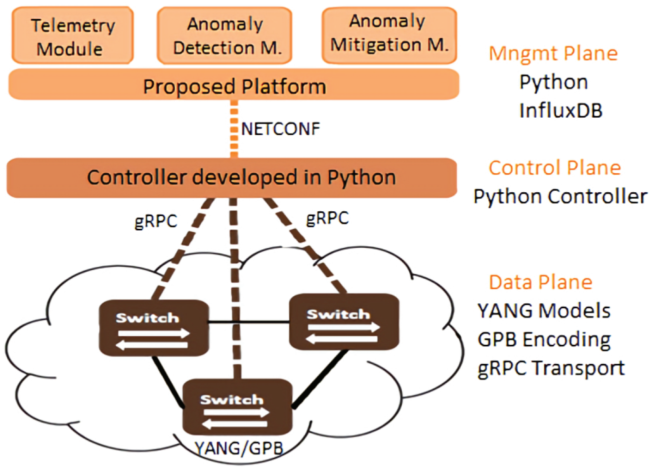


Figure 2. Protocol stack of the work in SDN arch.

Today, all IT processes are getting digitalized and the amounts of data generated by these systems reach very big sizes. One of the most common use cases is the collect-store-process [28] model to handle these massive amounts of data. In this model, dozens of different types of data collected from hundreds of different sources are accumulated by the collectors in a pipeline to be processed. Owing to the fact that we regard the anomaly detection and mitigation as a big data problem because of the high volume data streamed in the model-driven case, the collect-store-process model is combined with model-driven streaming telemetry to improve scalability in this study.

Our research shows that the success of real-time or near-real-time intrusion detection is directly associated with the fact that the monitoring method is able to respond at the same speed and accuracy. Flow collector type, flow data structure, pull or push mechanism, and sampling or event-based measurement methods are quite determinant attributes for anomaly detection success. Model-driven telemetry allows network devices to export very high dimensions and quantities of data out of the device in a formatted way that also provides high scalability. There are many collectors in the field but almost none of them support model-driven telemetry. In this paper, we present our developed and coded GPB-Compact encoding mechanism in an open-source collector as the collection part of the collect-store-process model. The details of the study are given in Section 2.1.1.

The second part of the collect-store-process model is storing the streamed data. The collectors subscribe to the metrics to take them out of the box in some seconds or even in some milliseconds in this approach. Time series databases (TBSDs) are developed for such subsecond task purposes. In contrast to traditional databases, TSDBs index the streamed data by time. This mechanism provides high-performance indexing, low-latency storage, and well-known APIs for the units where real-time streaming data will be processed [29]. Due to having a Python client, which allows users to apply their own code quickly on the database, micrometric measurement accuracy and in-RAM correlation features, InfluxDB is used as the TSDB in this paper.

The third part of the collect-store-process model is processing the streamed data stored in the TSDB. The Python [30] programming language is used to code modules and monitor the dashboard in this study.

2.1. Telemetry module

Streaming telemetry with a push mechanism is a novel technology that presents continuous, uninterrupted, scalable, and efficient communication by providing the operator a modeled data structure. This approach that provides fine-granular and high-frequency data about network states and other measurement statistics plays an important role in monitoring network events, performance, and security metrics. In order to increase the speed of the systems, it is more important how data are modeled and parsed rather than how data are transported [31]. Thus, we customized a collector to collect measurement data in an encoding mechanism that is commonly used in the literature for many purposes but is new in anomaly detection and mitigation.

2.1.1. Customized collector

While one of the basic functions is migration from polling to pushing mechanism, another innovation that provides scalability and flexibility in streaming telemetry are the data serialization/encoding abilities where data are converted into the desired format at the collector. Pipeline Collector is an open-source data bus that can carry model-based data generated by the IOS XR, which is the next-generation network operating system developed by Cisco. Besides the many advantageous features of Pipeline, our research shows that it does not support the GPB-Compact encoding protocol.

The data sizes of common encoding methods are given in Figure 3. As shown, GPB-Compact has minimum overhead and minimum capacity cost as it defines all data in binary format. Furthermore, GPB-Compact significantly affects the amount of streamed data, which directly influences the measurement cost. It provides maximum bandwidth efficiency, which also offers high scalability and high speed in data collection, which are crucial for the aims of this paper. On account of these reasons, the GPB-Compact encoding scheme is implemented to decrease measurement costs and increase speed in Pipeline Collector.

In order to determine the best combination of encoding method and transport protocol, a script is coded and performed. The algorithmic model of the script is given in Algorithm 1, and benchmark results are listed

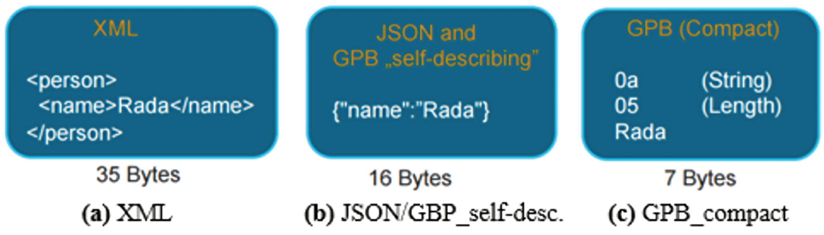


Figure 3. Capacity cost comparison of different encoding mechanisms.

in Table 1. The results show that the gRPC + GPB (Compact) combination minimizes the measurement cost in terms of produced data amount, efficiency in bandwidth consumption, speed, and reliability. As stated in their RFCs, TCP and gRPC are more resistant to packet loss than UDP. Thus, UDP's reliability is defined as "Low" while the others are "High". The K-means [32] clustering algorithm is used to define datasets as "Low", "Medium", and "High".

Considering all the reasons given above, the gRPC + GPB (Compact) combination is chosen for this study. The Go programming language is used for GPB (Compact) implementation.

Table 1. Performance comparison of the protocols.

Streaming transport protocol	Encoding	Data model	Data size for specific dataset (KB)	Bandwidth efficiency (low/med./hi.)	1KB data arrival time to the collector (ms)	Speed (1/latency) (low/med./hi.)	Reliability (low/high)
UDP	GPB (Comp.)	YANG	40	High	0.01895	High	Low
UDP	KVGPB	YANG	260	Med	0.06942	Med	Low
UDP	JSON	YANG	310	Low	0.79839	Low	Low
TCP	GPB (Comp.)	YANG	60	High	0.57751	Med	High
TCP	KVGPB	YANG	275	Med	0.85356	Low	High
TCP	JSON	YANG	360	Low	0.91675	Low	High
gRPC	GPB (Comp.)	YANG	45	High	0.02653	High	High
gRPC	KVGPB	YANG	240	Med	0.03163	High	High
gRPC	JSON	YANG	345	Low	0.06032	Med	High

2.1.2. Modeled data

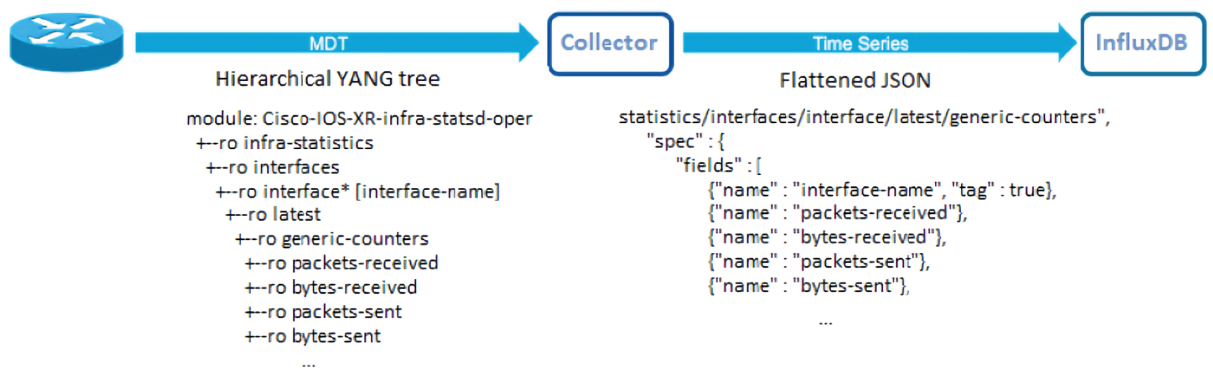
Although the YANG data modeling language offers a flexible, scalable, structural, and hierarchical framework by pushing measurement data from devices to collectors, the data must be converted into the flat format because time series databases can process data in a single layer. Thus, YANG-modeled measurement data in the hierarchical design must be converted into a structural but flat format while passing from collector to the InfluxDB. For this purpose, the measurement data are converted to JSON format as seen in Figure 4.

2.2. Anomaly detection module

Although there are many definitions related to anomalies in the literature [4–7], all expressions intersect in the common description of "unexpected behavior in usual traffic patterns". It is a critical issue to detect anomalies at the immediate time of the start of an abnormal pattern. The main objective of anomaly detection systems is the highest detection ratio of the correct anomaly in the shortest duration with the lowest count of false-positive alarms [33].

Algorithm 1 Performance comparison algorithm.

```
$dataSet$ = MeasurementDataExample
$dataModel$ = “YANG”
$transport$ = “UDP”, “TCP”, “gRPC”
$encoding$ = “GPB Compact”, “KVGPB”, “JSON”
$dataSetForArrival$ = 1KBDataSetExample
$Controller\_IP$ = “172.16.16.16”
procedure DataSet(dataSet, dataModel, transport, encoding, Controller_IP)
    $size\_transport \leftarrow transport.size$
    $size\_encoding \leftarrow encoding.size$
    $f() \leftarrow f(transmitdataingiven, dataModel, transport, encoding, Controller\_IP)$
    for $i < size\_transport$ do
        for $j < size\_encoding$ do
            $stream \leftarrow f(dataSet, dataModel, transport[i], encoding[j], Controller\_IP)$
            $measured.append \leftarrow stream.size$
            return $stream.size$
        end for
    end for
end procedure
procedure DataArrivalTime(dataSetForArrival, dataModel, transport, encoding, Controller_IP)
    $size\_transport \leftarrow transport.size$
    $size\_encoding \leftarrow encoding.size$
    $f() \leftarrow f(transmitdataingiven, dataModel, transportl, encoding, Controller\_IP)$
    for $i = 1..size\_transport$ do
        for $j = 1..size\_encoding$ do
            $timer.start()$
            $stream \leftarrow f(dataSetForArrival, dataModel, transport[i], encoding[j], Controller\_IP)$
            $timer.stop()$
            $measured2.append \leftarrow timer$
            return $timer$
        end for
    end for
end procedure
procedure LabelData(measured)
    $n\_clusters \leftarrow 3$
    $size\_measured \leftarrow measured.size$
    $labels \leftarrow [Low, Medium, High]$
    $kmeans \leftarrow KMeans(c\_clusters).fit(measured)$
    for $point$ in $kmeans$ do
        $cluster\_labels \leftarrow define\_label(kmeans, labels)$
    end for
    return $cluster\_labels$
end procedure
procedure BandwidthEfficiency(measured)
    return $LabelData(measured)$
end procedure
procedure Speed(measured)
    return $LabelData(measured2)$
end procedure
```



```
module: Cisco-IOS-XR-infra-statsd-oper
 +--ro infra-statistics
  +--ro interfaces
   +--ro interface* [interface-name]
    +--ro latest
     +--ro generic-counters
      +--ro packets-received
      +--ro bytes-received
      +--ro packets-sent
      +--ro bytes-sent
         ...
```

```
statistics/interfaces/interface/latest/generic-counters",
   "spec" : {
      "fields" : [
         {"name" : "interface-name", "tag" : true},
         {"name" : "packets-received"},
         {"name" : "bytes-received"},
         {"name" : "packets-sent"},
         {"name" : "bytes-sent"},
                  ...
```

Figure 4. Metric format conversion from YANG to JSON.

In this paper, telemetry data collected using a model-based streaming telemetry approach by the collector and stored as time series are analyzed in real time by a customized anomaly detection algorithm coded in Python. Holt's prediction algorithm is enhanced for better detection by developing adaptive error constant, service level control, and gradual activation mechanisms. Our experiments show the efficiency of the implementations in reducing false positives and increasing accuracy.

2.2.1. Adaptive error constant

One of the most common methods for detecting network anomalies is taking a model of the normal traffic pattern of the network, called the baseline, and capturing the deviations in this baseline curve. Holt's exponential smoothing algorithm is one of the widely used methods to forecast time series data. The method can produce successful results because of its features whereby the data trend can change randomly in time, its performance is correlated with big data, and it is applicable for both deterministic and stochastic series.

In the method, α is used for an error smoothing factor and β is used for trend estimation. Errors are smoothed using α, and they are added into the model. The model performs as the cumulative sum of a predetermined estimation of the previous period and a certain percentage of the previous period's error. If $\alpha = 0$, the previous period's error is not taken into account. If α approaches one, the error in the previous period affects prediction more intensively. The success of the estimation method is linked to the determination of the α coefficient that contains the lowest error. The α constant, which minimizes the error, is determined by trial in the current Holt's algorithm. Equations 1, 2, and 3 are used in Holt's algorithm to calculate the forecast in period t, the trend in period t, and the forecast in period t+1, respectively:

$$F_t = \alpha Y_{t-1} + (1-\alpha)(F_{t-1} - T_{t-1}), \tag{1}$$

$$T_t = \beta(Ft - F_{t-1}) + (1-\beta)T_{t-1}, \tag{2}$$

$$F_{t+1} = F_t + T_t, \tag{3}$$

where α is the smoothing constant for the level, β is for trend estimation, Y_t is the real value, F_t is the forecast value, and T_t is the trend value in the period t ($0 \leqslant \alpha, \beta \leqslant 1$).

Mean squared errors (MSE) and the mean of absolute deviation (MAD) are widely used techniques in the literature to figure out errors. The α value where the MSE is the smallest will make the most successful prediction. A script is developed to determine the best α that has minimum MSE and MAD by changing the α value from 0.1 to 0.9 for eight traffic patterns. Scapy[1] is implemented to generate arbitrary traffic patterns, and TCPreplay[2] is used to replay captured packets. Generated traffic patterns are counter values of 5-min in length observed every second from physical interfaces of the MDT device. They include different amounts of network traffics such as HTTP, FTP, DNS, SMTP, UDP, and ICMP. The traffic matrix changes randomly to simulate more realistic traffic patterns. After the generation process, one dataset looks like “74, 178, 350, 500, 850, 1100…” and another one looks like “300, 750, 1400, 1950, 2700, 3400…”

The script is given in algorithmic form in Algorithm 2, and the results are given in Table 2. The result shows that the best α with minimum error varies according to the dataset. Considering that the best α value varies according to the traffic pattern, the prerequired dynamic learning module is added to Holt’s exponential smoothing algorithm to predict the best α with minimum error that specifies the traffic. Dynamically calculated and adaptive to traffic patterns, the α constant is planned to be used after this learning process. In order to test the development, predefined/static α and adaptively learned α are applied to test traffic that includes 50 different patterns. In the case where the predefined/static α value is used, the success rate of detecting anomalies is 62%, and the success rate is 90% when the α value calculated adaptively in the prerequired dynamic learning process is used. It is shown that the prerequired dynamic learning module and the adaptive α constant have significant and positive effects on the success of the developed anomaly detection system.

For the β factor, values closer to zero provide more accurate results for the streaming telemetry datasets. In other words, as the β constant approaches one, the prediction system becomes unstable. Hence, β is defined as 0.1 in this paper.

Algorithm 2 Calculating MSE and MAD to find the best α value.

```
1: $alpha\_value \leftarrow 0.1$
2: $minMSE, minMAD, minAlpha \leftarrow 99999999$
3: procedure BEST_ALPHA(dataSets)
4:     for $dataset$ in $dataSets$ do
5:         for $count = 1..9$ do
6:             $detect \leftarrow HoltsDoubleExpSmoothing(dataset[0], alpha = alpha\_val, beta = 0.1)$
7:             $forecasted\_series \leftarrow detector.detect(dataset[1:])$
8:             $doubleseries \leftarrow [t[0]$ for $t$ in $[forecasted\_series] + [detector.forecasted]$
9:             $difference \leftarrow doubleseries[\mathbf{len}(doubleseries) - 1])(dataset[\mathbf{len}(dataset) - 1])$
10:            $MSE[count][1] \leftarrow MSE[count][1] + \mathbf{abs}(difference)$
11:            $MAD[count][1] \leftarrow MAD[count][1] + difference * difference$
12:            if $MSE[count][1] < minMSE$ then $\{minMSE \leftarrow MSE[count][1]; minAlpha \leftarrow alpha\_val\}$
13:            end if
14:            if $MAD[count][1] < minMAD$ then $\{minMAD \leftarrow MAD[count][1]; minAlpha \leftarrow alpha\_val\}$
15:            end if
16:            $alpha\_val \leftarrow alpha\_val + 0.1$
17:        end for
18:    end for
19:    return $minAlpha$
20: end procedure
```

[1]SCAPY (2017) Traffic Generator Tool [online]. Website https://github.com/secdev/scapy [accessed 10 Sep 2019].
[2]TCPreplay (2017). Traffic Replay Tool [online]. Website https://github.com/appneta/tcpreplay [accessed 10 Sep 2019].

Table 2. Differentiation of the best α value according to the traffic pattern.

Data-set	α	MSE	MAD	Data-set	α	MSE	MAD	Data-set	α	MSE	MAD	Data-set	α	MSE	MAD
	0.1	45.3120	2975.0832		0.1	202.5659	15955.8856		0.1	45.0739	789.8100		0.1	74.4233	3487.6433
	0.2	19.1651	555.3393		0.2	100.7790	3953.9221		0.2	20.3865	182.3752		0.2	79.0051	3753.1321
	0.3	18.4733	799.2909		0.3	101.5580	6036.4096		0.3	27.5024	458.2566		0.3	74.3822	3475.3415
	0.4	24.3542	1392.4823		0.4	110.0622	6224.3835		0.4	26.4567	334.5268		0.4	73.9062	3242.3256
1	0.5	25.8403	1211.2345	2	0.5	103.8075	5048.0674	3	0.5	28.3525	315.2422	4	0.5	81.2341	3974.3352
	0.6	25.9234	1028.2523		0.6	99.8884	4201.4497		0.6	20.3952	199.0525		0.6	84.2457	4214.6752
	0.7	26.5809	970.1804		0.7	101.6909	3870.1592		0.7	20.0122	172.2567		0.7	88.9421	4674.3241
	0.8	27.4140	975.3446		0.8	105.4525	3995.0424		0.8	36.2562	203.4525		0.8	78.2345	3845.3421
	0.9	28.0389	993.7642		0.9	109.2221	4845.9893		0.9	43.3425	662.5266		0.9	77.1345	3746.6352
	0.1	192.6747	1496.1279		0.1	373.9816	59658.2524		0.1	174.2456	18956.2562		0.1	134.2323	9247.4324
	0.2	93.9395	3551.2964		0.2	166.4965	12069.5626		0.2	173.5625	18600.6221		0.2	156.2445	9845.3453
	0.3	93.9732	5310.1592		0.3	216.9784	29549.5256		0.3	198.4221	23295.6621		0.3	170.1312	11345.256
	0.4	110.952	6652.4525		0.4	192.9656	21171.0425		0.4	203.4497	26733.5662		0.4	140.4215	9566.5245
5	0.5	102.0282	5028.0104	6	0.5	169.2158	13954.0525	7	0.5	207.4623	28906.2552	8	0.5	132.4555	9042.4273
	0.6	94.5284	3955.5256		0.6	169.9624	14902.4525		0.6	194.1134	25245.5266		0.6	141.4251	9765.3415
	0.7	93.6293	5503.6363		0.7	176.2318	11800.5798		0.7	180.5267	21456.5221		0.7	144.5764	10421.5256
	0.8	95.6293	3406.4903		0.8	183.5894	12058.8682		0.8	179.4198	21300.4986		0.8	160.4963	10567.352
	0.9	98.0608	3629.6546		0.9	189.5431	12272.0193		0.9	175.2455	20675.2345		0.9	154.5623	9703.1045

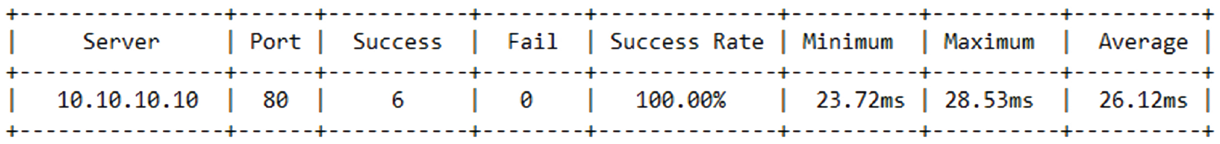
2.2.2. Service check module

In this study, it is aimed to decrease false-positive errors by checking whether the service is up or not. In cases where there is no attack for the hosts and the servers in the same local area network (LAN), the average latency of the communication is measured under 30.00 ms in many trials. It is observed that the packet loss ratio increases when there is an attack and service interruption. After checking the service quality at the moment of an attack, the final action is decided by analyzing whether the service is down or not. Algorithm 3 shows the pseudocode of the service check script, and Figure 5 shows the output of the service quality control code. It is seen from the experimental results that the service downtime increases exponentially if TCP packet count is increased and the false-positive alarms increase if TCP packet count is decreased. In order not to extend the service interruption and also to have the fewest false positives, the probing period is defined as six TCP packets. The packets are sent to the server's serving port that is 80 because of the HTTP in our scenario. The service success rate and the latency times are calculated as minimum, maximum, and average. The calculated success rate is utilized to decide the final action in the next module.

It is one of the most commonly faced challenges for anomaly detection and mitigation systems to generate alerts inaccurately. Anomaly detection algorithms can detect legitimate traffic as an intrusion. By developing a service check module, false-positive errors are mostly eliminated. The details are given in Section 3.2.

Algorithm 3 Service check algorithm.

```
1: procedure ServiceCheck(serverIP, Port)
2:     for $counter = 1..6$ do
3:         SendrandomlygeneratedTCPpackettotheserverIP:Port
4:         Calculate $latency, cumulative\_latency, packet\_loss$
       if $(packet\_loss <> 0)$ then $fail = fail + 1$ else $success = success + 1$ end if
5:     end for
6:     $success\_rate = (success/count) * 100$
7:     return success_rate, min($cumulative\_latency$), max($cumulative\_latency$), avg($cumulative\_latency$)
8: end procedure
```



```
+----------------+------+-----------+--------+--------------+----------+----------+----------+
|     Server     | Port |  Success  |  Fail  | Success Rate | Minimum  | Maximum  |  Average |
+----------------+------+-----------+--------+--------------+----------+----------+----------+
|  10.10.10.10   |  80  |     6     |   0    |   100.00%    |  23.72ms | 28.53ms  |  26.12ms |
+----------------+------+-----------+--------+--------------+----------+----------+----------+
```

Figure 5. Service quality control output.

2.2.3. Gradual activation

In this module, the network administrator enters the mitigation level (stated as *mitig_level* in the experiments) to specify the service cut-off level at which the anomaly prevention policies will be applied. This module provides significant flexibility and reliability for the networks that have various characteristics such as inconsistent, unstable, or bursty.

There are four mitigation levels as seen in Figure 6. The service success rate thresholds are defined empirically according to user experience under different levels of attacks. If the service success rate is 90% and above, it is decided that there is no attack, the alarm is produced inaccurately, the users are not affected, and no prevention policy will be applied. If the service success rate is between 70% and 90%, it is defined as low service interruption with only a few users experiencing connectivity issues to the service, while 50%–70% service success rate indicates high service interruptions with large numbers of users having connectivity issues. If the success rate is smaller than 50%, it means that more than half of the traffic drops and the service is no longer working. According to the degree of sensitivity determined by the network administrator, the anomaly prevention module is activated or not. Algorithm 4 shows the gradual activation code in the algorithmic model.

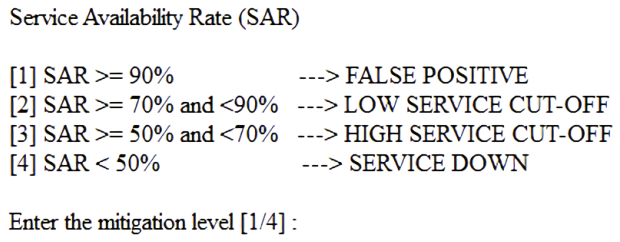
Algorithm 4 Gradual activation algorithm.

```
1: procedure GradualActivation(success_rate, mitig_level)
2:     if $success\_rate >= 90$ and $mitig\_level == 1$ then $Attack\_Prevention()$
3:     else if $success\_rate >= 70$ and $success\_rate < 90$ and $mitig\_level <= 2$ then $Attack\_Prevention()$
4:     else if $success\_rate >= 50$ and $success\_rate < 70$ and $mitig\_level <= 3$ then $Attack\_Prevention()$
5:     else if $success\_rate < 50$ and $mitig\_level <= 4$ then $Attack\_Prevention()$
6:     end if
7: end procedure
```

For instance, if the success rate is calculated as 90% and the mitigation level is defined as two or higher, the mitigation policy will not be applied. In another case, if the success rate is calculated as 50% and the mitigation level is defined as three or lower, the mitigation policy will be applied. The module offers useful predefinitions for admins to decrease false positives and increase adaptability to various traffic behaviors in advance.

2.2.4. Parsing attackers' IP addresses

After an anomaly is detected, attackers' IP addresses are parsed from the flow. The implemented parser sniffs the traffic and calculates the traffic matrix considering the volumes. The IP addresses that generate abnormally higher volume traffic to the server than the value predicted by the prediction algorithm are defined as attackers. The K-means algorithm is used to cluster normal and abnormal IP addresses. The output of the parser is given Figure 7.

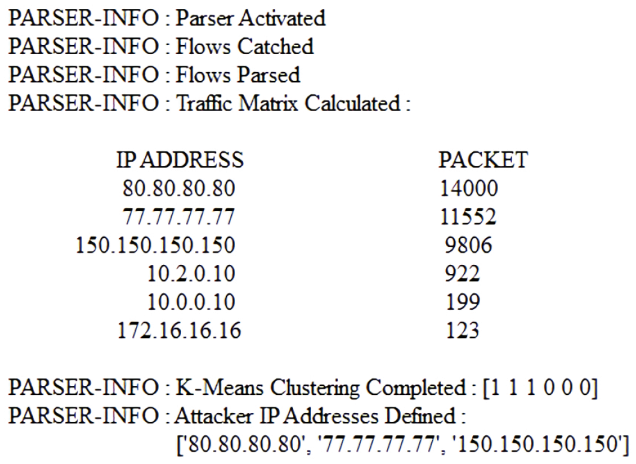


```
Service Availability Rate (SAR)

[1] SAR >= 90%                 ---> FALSE POSITIVE
[2] SAR >= 70% and <90%   ---> LOW SERVICE CUT-OFF
[3] SAR >= 50% and <70%   ---> HIGH SERVICE CUT-OFF
[4] SAR < 50%                   ---> SERVICE DOWN

Enter the mitigation level [1/4] :
```

Figure 6. Gradual activation admin panel.



```
PARSER-INFO : Parser Activated
PARSER-INFO : Flows Catched
PARSER-INFO : Flows Parsed
PARSER-INFO : Traffic Matrix Calculated :

                 IP ADDRESS                PACKET
                 80.80.80.80               14000
                 77.77.77.77               11552
            150.150.150.150                9806
                   10.2.0.10                922
                   10.0.0.10                199
                172.16.16.16                123

PARSER-INFO : K-Means Clustering Completed : [1 1 1 0 0 0]
PARSER-INFO : Attacker IP Addresses Defined :
                      ['80.80.80.80', '77.77.77.77', '150.150.150.150']
```

Figure 7. Defining attacker IP addresses by K-means clustering.

2.3. Anomaly mitigation module

If the protection module is activated as a result of gradual activation, the quarantine configuration for each attacker's IP address obtained from the flow is applied automatically without any manual trigger. The quarantine configuration can be of two types: i) reduction of the bandwidth of the attackers' IP addresses to a value that does not interfere with the network service availability anymore or ii) isolation of the attackers' IP addresses completely from the network. The quarantine configuration type is selected by the network admin. NETCONF is implemented for policy enforcement on network devices. Algorithm 5 presents the details of the quarantine algorithms for the proposed system.

3. Evaluation and test

In this module, we demonstrate how our real-time anomaly detection and mitigation platform with streaming telemetry concept works. We also present feature comparison covering the well-known works in the literature.

3.1. Experimental results

The volumetric attacks aim to make the service inaccessible by consuming the bandwidth and the other resources of the victim server by sending very high amounts of traffic. In our algorithm, high-volume anomalies such as UDP, TCP, or ICMP flood attacks can be detected in real time and with high accuracy. The test platform is deployed in a real networking infrastructure. All users, servers, and services are located in the Gazi University campus network that includes more than 1000 clients and dozens of virtual LANs (VLANs). Implemented logical and physical topologies are given in Figure 8 and Figure 9.

In the topology, IOS XRv 6.1.2 is used as the telemetry-supported network device. A Windows 2012 R2 server is deployed to provide DNS and HTTP services to the network. The collector, TSDB, process units, and SDN controller are located on one single virtual machine to minimize intersystem packet delay. Ubuntu 16.04.04 is used for this system and referred to as the "platform". InfluxDB 1.6.4 is used for the TSDB, while Python 2.7.12 is used for the coding platform. GNS3 is used to combine virtual machines in the topology.

Algorithm 5 Quarantine algorithms.

```
1: procedure MITIGATION(AttackerIPs, serverIP, mitigation_policy)
2:     Initiate YANG Models, RPC session for the MDT Device, NETCONF
         if $(mitigation\_policy == 1)$ $configfile = policy1.conf$ else $configfile = policy2.conf$ end if
3:     Connect Device using NETCONF
4:     Send $configfile$ using NETCONF
5: end procedure
6: policy1.conf
7: ipv4 access-list Prevention_Policy1
8: for x in AttackerIPs
9:   permit ipv4 host "+x+" host "+serverIP"
10: class-map attackers
11:   match access-group Prevention_Policy1
12: policy-map BandwidthReductionPolicy
13:   class attackers
14:     police 1000 1000 conform-action transmit exceed-action drop
15: interface g0/0/0/0
16:   service-policy output BandwidthReductionPolicy
17: policy2.conf
18: ipv4 access-list Prevention_Policy2
19: for x in AttackerIPs
20:   deny ipv4 host "+ x +" host "+ serverIP"
21: permit ipv4 any any
22: interface g0/0/0/0
23:   ipv4 access-group Prevention_Policy2 ingress
```

First of all, the gradual activation level is inserted by the network admin, and the adaptive α constant is calculated automatically from the streaming traffic in the learning mode. Then the network traffic is continuously monitored in protection mode. The anomaly detection module activates itself and starts the probing phase from the moment an anomaly is detected in the traffic. In order not to extend the service interruption and also to have the lowest rate of false positives, the probing period is defined as 5 s. When it is determined that the attack is continuing during this period, the next action proceeds according to the service availability threshold value and the selection of the gradual activation level. The service availability check query mostly takes 3-5 s. Finally, the mitigation mechanism is triggered and the attack is mitigated. In repeated tests, it is determined that the system recognizes the anomaly in 1 s and prevents it within an average of 8–10 s after the attack started.

Figure 10 and Figure 11 show the outputs of the developed real-time network monitoring dashboard before and during the attack. The blue line indicates the measurement data received from the device via streaming telemetry, and the orange line indicates the next data predicted by the prediction algorithm. In Figure 10, as there is a wide variety of traffic in the network, streamed and predicted data lines increase. The prediction algorithm defines the next value of the interface counter by its own baseline algorithm. In all new streamed data, the calculation repeats in real time. If there is an outlier or fluctuation in the streamed data as experienced between 12 and 15 s, the algorithm adapts itself dynamically unless there is an attack. In Figure 11, a real attack is applied to the HTTP/DNS server (IP address: 10.10.10.10) from a real PC (IP address: 10.2.0.10). In this case, the deviation between measured and estimated data increases. The algorithm tries to predict the next data by using streamed data. Thus, the predicted data grow correlated to the streamed data. After 5 s in

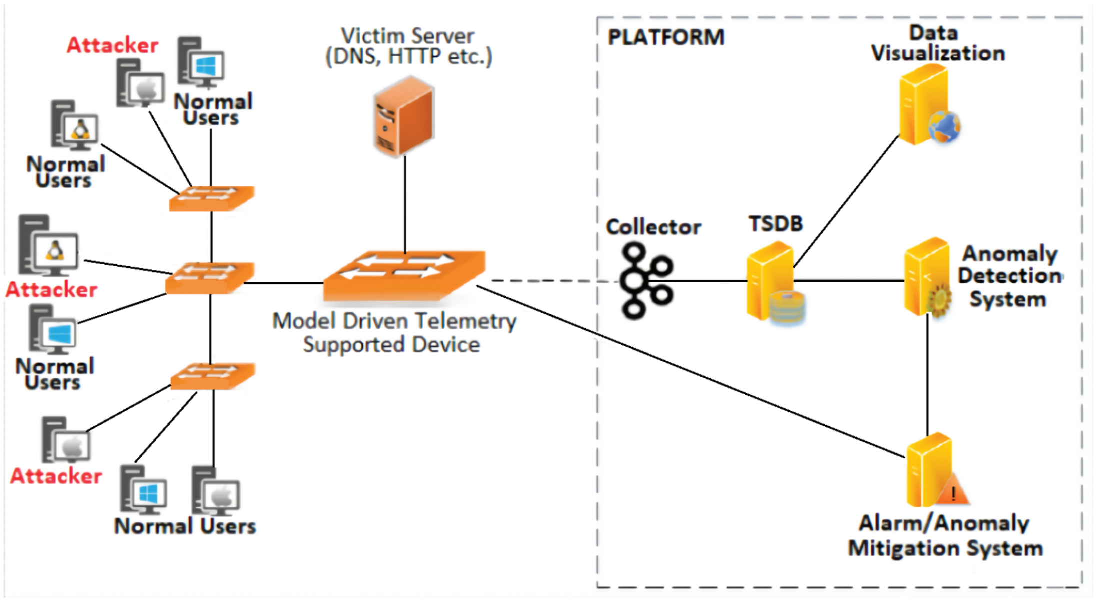


Figure 8. Logical topology.

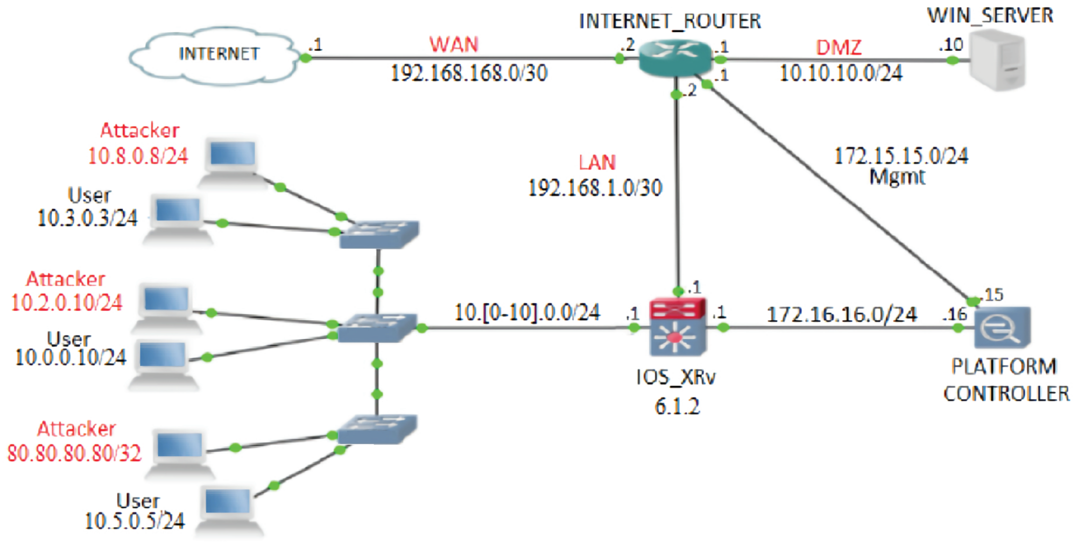


Figure 9. Physical topology.

the probing stage, if the streamed data volume is still higher than predicted, the developed platform perceives that there is an attack. The system reaches the next stage, where the service check and gradual activation modules become active, respectively. Approximately 10 s later, the anomaly mitigation module becomes active and the attack is mitigated by isolating the attackers' IP addresses from the network. Immediately after the mitigation policy is activated and attack is mitigated, normal users' communication with the server returns to normal within some milliseconds and the clients can reach the server.

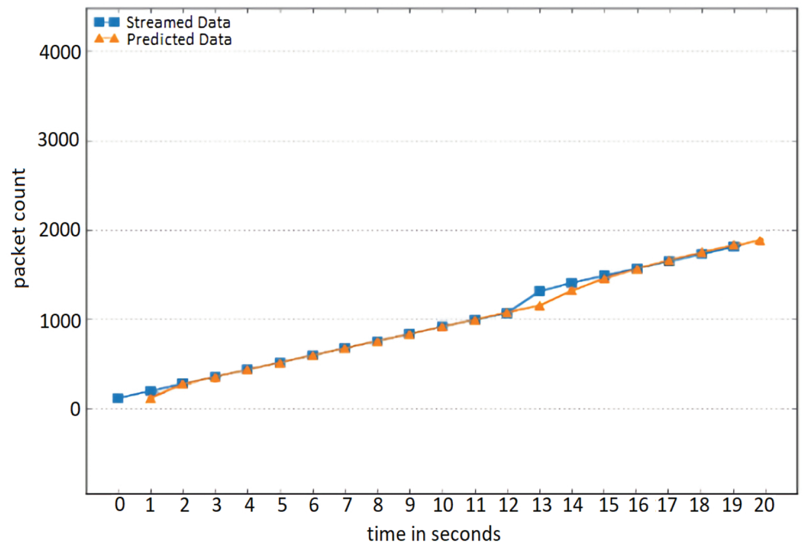


Figure 10. Dashboard output - legitimate traffic.

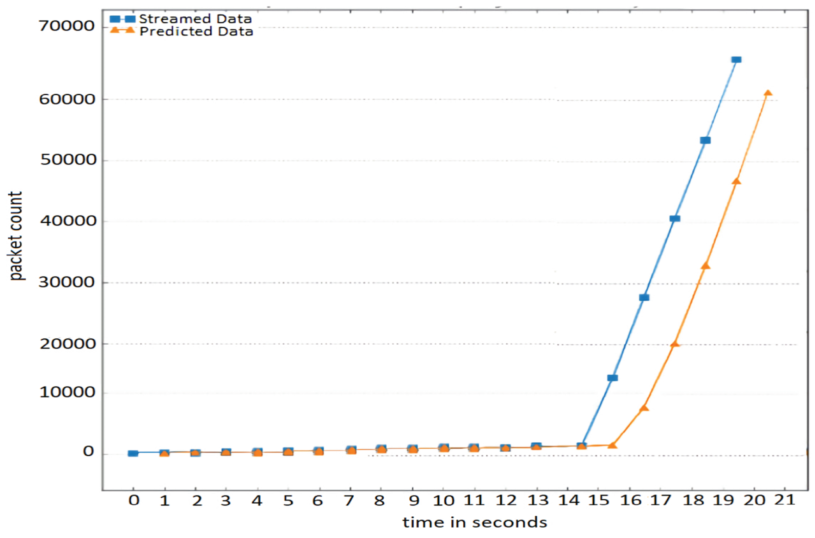


Figure 11. Dashboard output - abnormal traffic.

3.2. Performance tests

In order to evaluate our proposed platform, we implemented 100 different network patterns in the real Gazi University campus network. The Scapy tool is used to generate patterns in addition to campus network traffic. The patterns are 5-min packet counter values that are compositions of various network protocols such as HTTP, FTP, DNS, TCP-SYN, TCP-ACK, UDP, and ICMP. In some patterns, the protocol weights are intentionally altered during the 5-min period to create a wide range and more realistic traffic pattern space.

Seventy-five of 100 patterns include volumetric flood attacks on the given network protocols above and 25 of 100 patterns are legal, daily routine traffics of the campus. The Tcpreplay tool is used to capture generated patterns to use them in other performance tests. Some of the streaming telemetry datasets used in the experiments are listed in Table 3.

Table 3. Dataset examples used in the tests.

Dataset1	92, 510, 980, 1320, 1900, 2310, 2740, 3178, 3512...
Dataset2	1400, 2950, 4250, 6750, 7800, 9300, 10620, 11480...
Dataset3	6500, 16000, 23000, 32500, 51000, 60000, 67000...

Sixty-seven of the 75 attack traffics are detected correctly. All 25 legitimate patterns are marked as normal. Equations 4 and 5 are used to calculate the detection rate (DR) and false positive rate (FPR). Equations 6, 7, 8, and 9 are used to calculate accuracy, precision, recall, and F1 scores, respectively. Table 4 presents the accuracy performance results of the proposed platform.

$$DR = \frac{TP}{TP + FN}, \tag{4}$$

$$FPR = \frac{FP}{TN + FP}, \tag{5}$$

$$Accuracy = \frac{TP + TN}{TP + FP + FN + TN}, \tag{6}$$

$$Precision = \frac{TP}{TP + FP}, \tag{7}$$

$$Recall = \frac{TP}{TP + FN}, \tag{8}$$

$$F1 = 2 * \frac{Precision * Recall}{Precision + Recall}, \tag{9}$$

where TPs (true positives) are attack patterns predicted as attacks, FNs (false negatives) are attack patterns predicted as legitimate, FPs (false positives) are legitimate patterns predicted as attacks, and TNs (true negatives) are legitimate patterns predicted as legitimate.

As seen from Table 4, the platform has no mitigation action for legitimate traffic. By developing a false-positive robustness module, the developed system provides high accuracy in false-positive performance. The overall success rate of the normal and abnormal traffic detection of the developed system is 92%.

Table 4. Accuracy calculation results of the proposed platform.

Datasets	Detection rate (%)	False positive rate (%)	Accuracy	Precision	Recall	F1
Abnormal datasets	89	11	-	-	-	-
Legitimate datasets	100	0	-	-	-	-
Total	92	8	0.92	1	0.89	0.94

The datasets are classified into attack types and they are implemented to measure the real-time anomaly detection ability of the proposed platform. In every attack type, 10 datasets are selected and implemented. The average detection times are calculated and presented in Table 5. As is seen, all attack types are detected in under 1 s. UDP-based attacks are detected faster than TCP attacks.

In order to evaluate the granular attack detection ability, short-period attacks are implemented. The attack periods are gradually decreased. The proposed system can easily detect the attacks that last more than

0.7 s. The attacks lasting between 0.7 and 0.4 s are not detected stably. The system can not sense attacks lasting less than 0.4 s. The results are given in Table 6.

Table 5. Attack detection time tests.

Attack type	Avg. detection time (s)
TCP-SYN flood	0.948
TCP-ACK flood	0.989
UDP flood	0.772
HTTP flood	0.874
ICMP flood	0.831

Table 6. Granular attack detection.

Attack period (s)	Detection
61–300	Successful
31–60	Successful
30–1	Successful
1	Successful
1–0.70	Successful
0.69–0.40	Unstable
0.39–0.00	Failed

In order to evaluate the CPU overhead of the proposed platform, the CPU resources of the controller when anomaly detection and mitigation Python codes are run are observed before and during the process. After the interface metrics of the device in the data plane are started to be streamed every second starting from the 40th second, only 2% CPU fluctuations are observed. When the utilized bandwidth is incremented gradually, the CPU load is not affected by the bandwidth (BW) as seen in Figure 12. As the streaming telemetry concept promises, the system performance is not affected by how much data flows through it. In this way, the concept provides highly scalable measurement infrastructures for multitenant environments or big networks with many devices. Large amounts of metrics from many devices can be monitored and fetched without any performance issues. In this way, the proposed work provides high scalability.

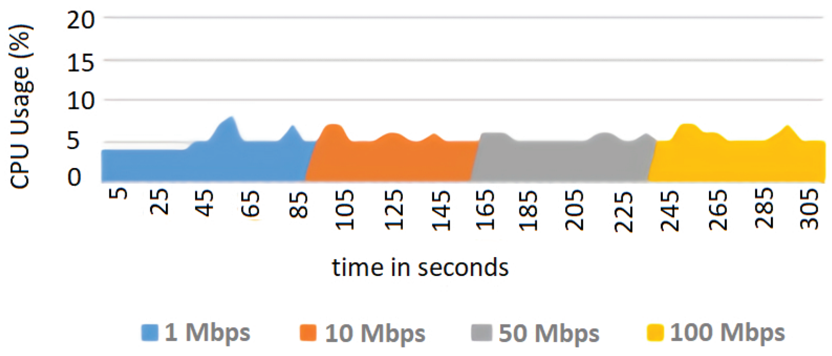


Figure 12. CPU overhead under diff. BWs.

3.3. Comparison of results

The feature comparison table of the common anomaly detection and mitigation systems in the literature is given in Table 7. The same datasets and experimental setups are used to calculate results. The studies are investigated with three main approaches.

In the first approach, it is seen that the attack detection accuracy rates are low as expected because of the fact that sampling-based mechanisms are relatively strong to sense granular attacks. The studies also

Table 7. Comparison of the approaches.

	Sampling and OF-based appr.			Only OpenFlow-based appr.		Streaming telemetry-based appr.		
Features	Giotis[5]	FlowTr.[7]	OpenW.[14]	LightW.[16]	Mehdi et al.[17]	Cheng[22]	DenStr.[24]	Proposed Work
False positive ratio	39.3%	#NA	#NA	0.59%	30%	22%	0.4%	11%
Robustness to false positives	No	No	No	No	No	No	Yes	Yes
Attack detection period	30 s	48 s	60 s	26 s	23 s	12 s	16 s	1 s
Attack mitigation period	30 s	10 s	120 s	No mitigation	#NA	20 s	30 s	10 s
Granular attack detect. (attacks less than 30 s)	No	Yes	No	Yes	No	Yes	Yes	Yes
CPU Overhead	25%	19%	20%	#NA	17.54%	#NA	#NA	2%
Accuracy	69%	63%	#NA	98.61%	90%	79%	99%	92%

do not have any protection mechanisms for false-positive alarms. In addition, they have higher overheads and computational costs than the other two approaches.

The studies in the second approach use only OpenFlow to get measurement metrics out of the devices. Thus, they have higher accuracy than sampling-based studies. However, OpenFlow-only studies are mainly used for anomaly detection in a passive manner. The anomaly mitigation abilities are missing. Attack detection periods are lower than those for sampling-based techniques, but they are still higher when compared to streaming telemetry-based approaches.

In the literature, streaming telemetry-based approaches are getting more popular because of their scalability, integration ability, and programmability features. As they consider all metrics flows through the network, granular attack detection is easily possible with this approach. Because of its modeled data structure and metric subscription architecture, it is highly flexible and scalable to integrate third-party solutions and gain visibility of network behaviors in a very short time. When streaming telemetry-based approaches are analyzed, our proposed platform provides lower false-positive protection with real-time anomaly detection and mitigation abilities. By using the model-driven streaming telemetry architecture with our own improvements, the proposed platform provides near real-time protection. It can detect granular and also short-term attacks in 1 s and mitigate them in 10 s. By improving Holt's algorithm, it provides augmented accuracy, flexible adaptation, and high customization for the traffic and network administrator's needs.

4. Conclusion

In this paper, a real-time anomaly detection and mitigation system is proposed on SDN architecture. The developed system combines model-driven streaming telemetry and an enhanced version of Holt's double exponential smoothing forecasting algorithm. The YANG data model, GPB (Compact) encoding method, and gRPC transport model are implemented in the telemetry module, and Holt's forecasting algorithm is enhanced with adaptive error constant, service check module, and gradual activation features for the anomaly detection module. NETCONF is used to enforce predefined prevention policies on the model-driven telemetry device for the anomaly mitigation module. The experiments are conducted on the Gazi University campus network. The results indicate that our platform detects volumetric anomalies in real time and with high accuracy.

In this study, only volumetric attacks are considered. In further studies, nonvolumetric but application-specific and sophisticated attacks can also be investigated due to emerging and changing attack patterns. On the other hand, as well as the SDN architecture that provides a wide view of network and programmability abilities by omitting the data plane from the control plane, the centralized network functions are more vulnerable to attacks than before. Thus, other kinds of approaches like artificial intelligence and machine learning algorithms might be used to improve protection capability and accuracy.

References

[1] Bawany NZ, Shamsi JA, Salah K. DDoS attack detection and mitigation using SDN: methods, practices, and solutions. Arabian Journal for Science and Engineering 2017; 42 (2): 1-17.

[2] Yu M, Jose L, Miao R. Software Defined Traffic Measurement with OpenSketch. Berkeley, CA, USA: Networked Systems Design and Implementation; 2013. pp. 29-42.

[3] Giotis K, Androulidakis G, Maglaris V. Leveraging SDN for efficient anomaly detection and mitigation on legacy networks. In: 3rd European Workshop on Software Defined Networks; Budapest, Hungary; 2014. pp. 85-90. doi: 10.1109/EWSDN.2014.24

[4] Wang P, Chao KM, Lin HC, Lin WH, Lo CC. An efficient flow control approach for SDN-based network threat detection and migration using support vector machine. In: IEEE 13th International Conference on e-Business Engineering; Macau; 2016. pp. 56-63. doi: 10.1109/ICEBE.2016.020

[5] Giotis K, Argyropoulos C, Androulidakis G, Kalogeras D, Maglaris V. Combining OpenFlow and sFlow for an effective and scalable anomaly detection and mitigation mechanism on SDN environments. Computer Networks 2014; 62 (1): 122-136. doi: 10.1016/j.bjp.2013.10.014

[6] Rehman SU, Song WC, Kang M. Network-wide traffic visibility in OF@TEIN SDN testbed using sFlow. In: 16th Asia-Pacific Network Operations and Management Symposium; Hsinchu, Taiwan; 2014. pp. 1-6. doi: 10.1109/APNOMS.2014.6996541

[7] Buragohain C, Medhi N. FlowTrApp: An SDN based architecture for DDoS attack detection and mitigation in data centers. In: 3rd International Conference on Signal Processing and Integrated Networks; Noida, India; 2016. pp. 519-524. doi: 10.1109/SPIN.2016.7566750

[8] Jose L, Yu M, Rexford J. Online measurement of large traffic aggregates on commodity switches. In: Proceedings of the USENIX HotICE; Boston, MA, USA; 2011. pp. 13-14.

[9] Chowdhury S, Bari M, Ahmed R, Boutaba R. PayLess: A low cost network monitoring framework for Software Defined Networks. In: Proceedings of the 14th IEEE/IFIP NOMS; Krakow, Poland; 2014. pp. 1–9. doi: 10.1109/NOMS.2014.6838227

[10] Yu C, Lumezanu C, Zhang Y, Singh V, Jiang G et al. FlowSense: Monitoring network utilization with zero measurement cost. In: Proceedings of the 14th IPAM; Toronto, Canada; 2013. pp. 31–41. doi: 10.1007/978-3-642-36516-4_4

[11] Wang MH, Wu SY, Yen LH, Tseng CC. PathMon: Path-specific traffic monitoring in OpenFlow-enabled networks. In: 8th International Conference on Ubiquitous and Future Networks; Vienna, Austria; 2016. pp. 775-780. doi: 10.1109/ICUFN.2016.7537143

[12] Ballard JR, Rae I, Akella A. Extensible and scalable network monitoring using OpenSAFE. In: Proceedings of the 2010 Internet Network Management Conference on Research on Enterprise Networking; Berkeley, CA, USA; 2010. p. 8.

[13] Khan F, Hosein N, Ghiasi S, Chuah CN, Sharma P. Streaming solutions for fine-grained network traffic measurements and analysis. IEEE/ACM Transactions on Networking 2014; 22 (2): 377-390. doi: 10.1109/TNET.2013.2263228

[14] Zhang Y. An adaptive flow counting method for anomaly detection in SDN. In: Proceedings of the 9th ACM CoNEXT; California, USA; 2013. pp. 25–30. doi: 10.1145/2535372.2535411

[15] Shirali S, Ganjali Y. Efficient implementation of security applications in OpenFlow controller with FleXam. In: IEEE 21st Annual Symposium on High-Performance Interconnects; San Jose, CA, USA; 2013. pp. 49-54. doi: 10.1109/HOTI.2013.17

[16] Braga R, Mota R, Passito A. Lightweight DDoS flooding attack detection using NOX/OpenFlow. In: Proceedings of the 35th Conference on Local Computers; Denver, CO, USA; 2010. pp. 408-415. doi: 10.1109/LCN.2010.5735752

[17] Mehdi SA, Khalid J, Khayam SA. Revisiting traffic anomaly detection using software defined networking. In: Proceedings of the 14th International Conference on Recent Advances in Intrusion Detection; Menlo Park, CA, USA; 2011. pp. 161-180.

[18] He D, Chan S, Ni X, Guizani M. Software-defined-networking-enabled traffic anomaly detection and mitigation. IEEE Internet of Things Journal 2017; 4 (6): 1890-1898. doi: 10.1109/JIOT.2017.2694702

[19] Vasilomanolakis E, Karuppayah S, Mühlhauser M, Fischer M. Taxonomy and survey of collaborative intrusion detection. ACM Computing Surveys 2015; 47 (4): 1-33. doi: 10.1145/2716260

[20] Khairi MH, Sharifah HS, Abdul NM, Abdullah AS, Hassan MK. A review of anomaly detection techniques and distributed denial of service (DDoS) on software defined network (SDN). Engineering, Technology & Applied Science Research 2018; 8 (2): 2724-2730.

[21] Miller Z, Deitrick W, Hu W. Anomalous network packet detection using data stream mining. Journal of Information Security 2017; 2 (4): 158–168.

[22] Cheng J, Xu R, Tang X, Sheng VS, Cai C. An abnormal network flow feature sequence prediction approach for DDoS attacks detection in big data environment. Computers, Materials and Continua 2018; 55 (1): 95-119.

[23] Nataf E, Festor O. End-to-end YANG-based configuration management. In: IEEE Network Operations and Management Symposium; Osaka, Japan; 2010. pp. 674-684. doi: 10.1109/NOMS.2010.5488381

[24] Putina A, Rossi D, Bifet A, Barth S, Pletcher D et al. Telemetry-based stream-learning of BGP anomalies. In: Proceedings of the 2018 Workshop on Big Data Analytics and Machine Learning for Data Communication Networks; New York, NY, USA; 2018. pp. 15-20. doi: 10.1145/3229607.3229611

[25] Manso P, Moura J, Serrão C. SDN-based intrusion detection system for early detection and mitigation of DDoS attacks. Information 2019; 10 (3): 106. doi: 10.3390/info10030106

[26] McKeown N, Anderson T, Balakrishnan H, Parulkar G, Petrson L et al. OpenFlow: Enabling innovation in campus networks. ACM SIGCOMM Computer Communication 2008; 38 (2): 69-74. doi: 10.1145/1355734.1355746

[27] Holt CC. Forecasting seasonals and trends by exponentially weighted moving averages. International Journal of Forecasting 2004; 20 (1): 5-10. doi: 10.1016/j.ijforecast.2003.09.015

[28] Eliseev D, Farkhadov M. Modern methods to collect, store, and process big data in large-scale systems. In: 5th International Conference on Control, Instrumentation, and Automation; Shiraz, Iran; 2017. pp. 179-182. doi: 10.1109/ICCIAutom.2017.8258674

[29] Miller Z, Dickinson B, Deitrick W, Hu W, Wang AH. Twitter spammer detection using data stream clustering. Information Sciences 2014; 260: 64–73. doi: 10.1016/j.ins.2013.11.016

[30] Dobesova Z. Programming language Python for data processing. In: International Conference on Electrical and Control Engineering; Yichang, China; 2011. pp. 4866-4869. doi: 10.1109/ICECENG.2011.6057428

[31] Garima S, Rani S. Review on time series databases and recent research trends in time series mining. In: 5th International Conference - Confluence, The Next Generation Information Technology Summit; Noida, India; 2014. pp. 109-115. doi: 10.1109/CONFLUENCE.2014.6949290

[32] MacQueen J. Some methods for classification and analysis of multivariate observations. In: Proceedings of the Berkeley Symposium on Mathematical Statistics and Probability; Berkeley, CA, USA; 1967. pp. 281-297.

[33] Ahmed M, Mahmood N, Hu J. A survey of network anomaly detection techniques. Journal of Network and Computer Applications 2016; 60: 19–31. doi: 10.1016/j.jnca.2015.11.016