# Towards Real-Time Intrusion Detection for NetFlow and IPFIX

Rick Hofstede*, Václav Bartoš†, Anna Sperotto*, Aiko Pras*

* Design and Analysis of Communication Systems (DACS), Centre for Telematics and Information Technology (CTIT)
University of Twente, Enschede, The Netherlands
{r.j.hofstede, a.sperotto, a.pras}@utwente.nl
† IT4Innovations Centre of Excellence
Brno University of Technology, Brno, Czech Republic
ibartosv@fit.vutbr.cz

***Abstract*—DDoS attacks bring serious economic and technical damage to networks and enterprises. Timely detection and mitigation are therefore of great importance. However, when flow monitoring systems are used for intrusion detection, as it is often the case in campus, enterprise and backbone networks, timely data analysis is constrained by the architecture of NetFlow and IPFIX. In their current architecture, the analysis is performed after certain timeouts, which generally delays the intrusion detection for several minutes. This paper presents a functional extension for both NetFlow and IPFIX flow exporters, to allow for timely intrusion detection and mitigation of large flooding attacks. The contribution of this paper is threefold. First, we integrate a lightweight intrusion detection module into a flow exporter, which moves detection closer to the traffic observation point. Second, our approach mitigates attacks in near real-time by instructing firewalls to filter malicious traffic. Third, we filter flow data of malicious traffic to prevent flow collectors from overload. We validate our approach by means of a prototype that has been deployed on a backbone link of the Czech national research and education network CESNET.**

***Index Terms*—Internet measurements, Denial of service, Intrusion detection, NetFlow, IPFIX, Flow monitoring.**

## I. Introduction

Massive DDoS attacks are starting to become a new type of warfare, in which networks and servers are overwhelmed by network traffic. For example, the Spamhaus Project [1] has been targeted by attacks of more than 300 Gbps of bandwidth in early 2013 [2], large enough to overload Internet exchanges [3]. Other large flooding attacks that gained media attention were targeted at U.S. financial institutions in late 2012, where compromised Web servers were used as bots for creating a mixture of high-volume TCP, UDP, ICMP and other IP-based traffic [4]. All these DDoS attacks are usually volume-based, and therefore suitable for detection in a flow-based manner.

Flow export technologies, such as NetFlow [5] and IPFIX [6], provide a means for monitoring high-speed networks in a passive and scalable manner by aggregating packets into flows. Flows are defined in [7] as sets of IP packets passing an observation point in the network during a certain time interval, such that all packets belonging to a particular flow have a set of common properties. A typical deployment of flow export technologies is shown in Fig. 1. Flow exporters

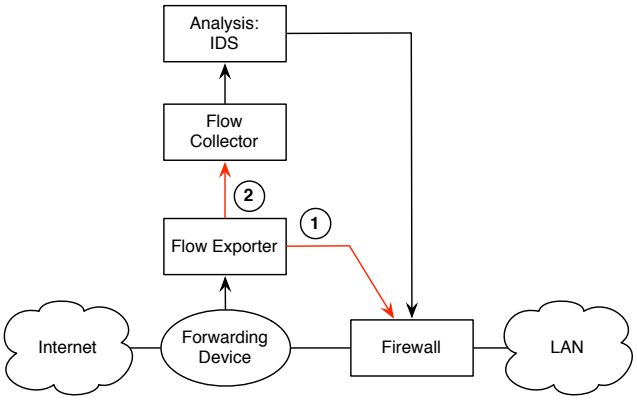
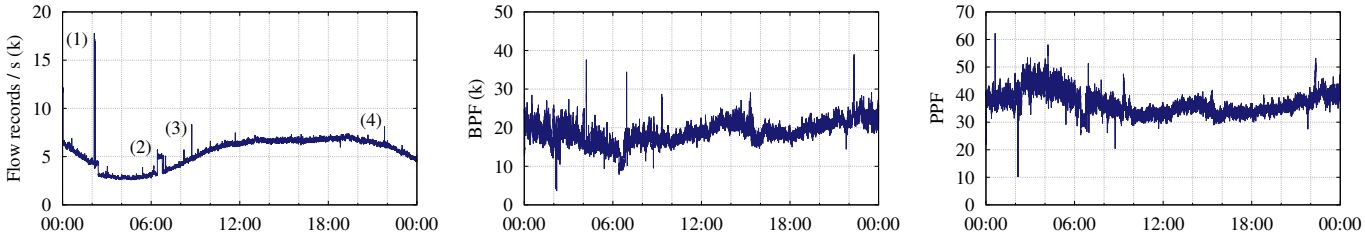



Fig. 1: Typical flow monitoring system deployment.

receive raw packets and aggregate them into flows, which is commonly referred to as *flow metering*. They can be part of forwarding devices (*e.g.* switches and routers), or separate devices that are dedicated to the task of flow export, as shown in Fig. 1. After a flow is considered to have terminated, flow data is exported to flow collectors for storage and pre-processing. Finally, analysis applications, such as intrusion detection systems (IDSs), retrieve flow data and analyze it [8]–[10], and potentially send out alerts or instruct firewalls.

Due to the design of NetFlow and IPFIX, flow-based IDSs are subject to delays during flow metering and collection [11]. These delays are in particular a consequence of timeouts for expiring flow records as part of the flow metering process, and processing times of flow collectors. Considering the default idle timeout applied by several vendors and the processing times of state-of-the-art flow collectors, this usually results in IDS detection delays in the order of minutes. Nevertheless, it is important to detect and mitigate as early as possible to limit the potential damage brought by large attacks, such as device overload and link capacity exhaustion. Our intuition tells us that moving parts of the detection process closer to the source may reduce detection times drastically. Given that a timely detection allows for timely mitigation, we propose a functional




(a) Flow record creations (b) Bytes per flow (BPF) (c) Packets per flow (PPF)

Fig. 2: Flow-Based DDoS Attack Metrics.

extension for flow exporters that integrates intrusion detection into the flow metering process. This avoids the delays incurred in typical flow monitoring systems and has the following advantages:

1) Mitigation of DDoS flooding attacks by blocking malicious traffic before it reaches the LAN (as illustrated by (1) in Fig. 1).
2) Mitigation of DDoS flooding attacks by filtering malicious flow data before it reaches and potentially overloads a flow collector (as illustrated by (2) in Fig. 1). We know from our operational experience that DDoS attacks often cause flow data loss due to collector overload [12]. Moreover, European backbone operators have also confirmed this problem in discussions we had with them.

Typical values for the idle timeout applied for expiring flow records range from 15 seconds (default value applied in Cisco's IOS [13]) to 60 seconds (default value applied in Juniper's Junos [14]). In addition, flow collectors often work based on time slots, which causes flow data to become available to analysis application only after the next time slot has started. For example, the state-of-the-art flow collector NfSen uses time slots of 5 minutes, resulting in an average delay of 150 seconds (2.5 minutes). The average delay between the moment in which a packet is metered and the time at which flow data is made available to analysis applications, is therefore at least 165 seconds, considering an idle timeout of 15 seconds. In this paper, we analyze whether our approach can reduce the delay up to 10% of this value. Besides this requirement, we target an intrusion detection module that is lightweight, having a minimal performance footprint of 10% in terms of CPU usage and memory consumption on a flow exporter. This is important since exporter operation is considered time-critical. Last, the accuracy of our intrusion detection module should be high enough, to ascertain a low number of false positives/negatives.

This paper is structured as follows. We start by analyzing flow-level characteristics of flooding attacks, mainly considering metrics that can be monitored in a lightweight manner (*i.e.* with a minimal performance footprint) on a flow exporter. Based on these findings, which are presented in Section II, we study existing flooding attack detection algorithms that can be modified to support the previously found metrics. These algorithms are presented in Section III. In Section IV, we present both the prototype that is used to validate whether all identified requirements have been fulfilled, and the utilized datasets. Validation results are presented in Section V, together with an example of the prototype in operation on a backbone link. The feasibility of this work in terms of deployability on various high-end forwarding platforms is discussed in Section VI. Finally, we draw our conclusions in Section VII.

## II. DDoS Attack Metrics

Flooding attacks are a type of (D)DoS attack that aim at exhausting targets' resources by overloading them with large amounts of traffic or (incomplete) connection attempts. As every connection attempt uses a different source port number and therefore results in a new flow, large numbers of flow records are exported to flow collectors, effectively canceling out the data aggregation advantage provided by flow export technologies. In case a target replies to a connection attempt, two flow records are exported per attempt. The same characteristics may apply to (large) network scans. Flow collectors need to process all resulting records, consisting of only a few packets and bytes, which may be more than they can handle.

In this section, we analyze which flow-level traffic metrics are suitable for lightweight detection of attacks on a flow exporter. Sadre *et al*. have previously identified four traffic metrics that change significantly during a DDoS flooding attack: flow record creations per second, average flow duration, average number of bytes per flow, and average number of packets per flow [12]. All but the average flow duration can be monitored on a flow exporter by using only counters, without the need to access and process each individual flow record after expiration. This makes these metrics particularly interesting for this work, in which we aim at designing a lightweight intrusion detection module for detecting large flooding attacks.

Time-series of the three considered metrics are shown in Fig. 2. The subfigures show data from one of the backbone links of the Czech national research and education network CESNET in November 2012. The number of flow record creations per second is shown in Fig. 2a. The diurnal pattern is clearly identifiable and several peaks can be observed. Given the flow-level characteristics of flooding attacks, we assume the peaks labeled with a number to indicate the presence of such attacks. We have validated this assumption by manually verifying the presence of scanning and flooding attacks in the flow data.

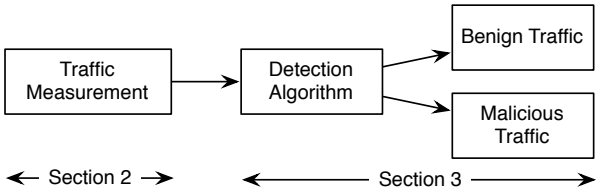



Fig. 3: Detection Workflow.

The number of bytes per flow (BPF) and packets per flow (PPF) are shown in Fig. 2b and Fig. 2c, respectively. Although the attacks identified in Fig. 2a can be observed as negative peaks here as well, the figures also show many other peaks, which make these metrics noisy. What can be confirmed from these figures, however, is that the attacks identified in Fig. 2a consist of many small flows with few and small packets.

Out of the three presented metrics in Fig. 2, the number of flow record creations (Fig. 2a) appears to be the most suitable metric for our purposes for the following reasons. First, it shows the least amount of noise, peaks are clearly identifiable, and the identified peaks have been confirmed to be attacks, as substantiated by the other metrics. Second, this metric is the best to fulfill the requirement of being lightweight, as only a single counter is needed that has to be reset after every measurement interval.

Although we have shown only day-long time-series in Fig. 2, we have verified that our conclusions are valid for the whole dataset. We will therefore use the number of flow record creations for our traffic measurements, as shown in Fig. 3. These measurements are performed in a continuous fashion and used as input for a detection algorithm, which on its turn classifies a measurement (sample) as benign or malicious. Two detection algorithms are discussed in the next section.

## III. Detection Algorithms

In this paper we consider an anomaly-based intrusion detection approach. One method for performing anomaly detection based on the analysis of time-series is forecasting, which uses previous measurements for forecasting the next value. If the measured value does not lie within a certain range of the forecasted value, a measurement sample is considered malicious and an anomaly has been detected. In this paper, we consider the following two algorithms:

**Algorithm 1**: Exponentially weighted moving average (EWMA) for mean calculation, extended by thresholds and a cumulative sum (CUSUM) [15]. We consider this our basic algorithm.

**Algorithm 2**: Algorithm 1, extended by seasonality modeling.

Although the second algorithm may intuitively be considered more accurate, it is likely to have a larger performance footprint in terms of memory consumption and processing complexity than the first algorithm, since more data needs to be stored and processed. Whether the larger performance footprint justifies the use of Algorithm 2 in terms of accuracy, will be investigated later in this paper.

Both algorithms rely on EWMA for calculating the mean over past values, which we use for forecasting the next value. Previous works have shown that EWMA can be used for anomaly detection (*e.g.* [15], [16]). It is defined as follows:

$$\bar{x}_t = \alpha \cdot x_t + (1-\alpha) \cdot \bar{x}_{t-1} \quad (1)$$

$$\hat{x}_{t+1} = \bar{x}_t \ , \quad (2)$$

where $x_t$ is the measured value, $\bar{x}_t$ is the weighted mean over current and past values at time $t$, $\hat{x}_{t+1}$ the value forecasted for time $t+1$, and $\alpha \in (0,1)$ a parameter which determines the rate in which previous values are discarded. When the value of $x_t$ becomes known, both the forecasting error $e_t$ and an upper threshold $T_{upper,t}$ can be calculated:

$$e_t = x_t - \hat{x}_t \quad (3)$$

$$T_{upper,t} = \hat{x}_t + max(c_{threshold} \cdot \sigma_{e,t}, M_{min}) \ , \quad (4)$$

where $c_{threshold}$ is a constant and $\sigma_{e,t}$ the standard deviation of previous forecasting errors. $M_{min}$ is a margin that is added to the measurement $\hat{x}_t$ to avoid instability in case $c_{threshold} \cdot \sigma_{e,t}$ is small. This solution prevents small peaks during quiet periods to be considered anomalous. Note that we do only consider an upper threshold and no lower threshold, since flooding attacks result, by definition, in a greater input value in terms of flow record creations than the forecasted value (as discussed in Section II).

Reporting an anomaly every time the upper threshold has been exceeded might cause a large number of false positives. To overcome this problem, we use a cumulative sum (CUSUM), which is widely used in anomaly detection algorithms [15], [17], [18]. The differences between the measurement and the upper threshold are summed ($S_t$), and an anomaly is detected when the sum exceeds threshold $T_{cusum,t}$:

$$S_t = max(S_{t-1} + (x_t - T_{upper,t}), 0) \quad (5)$$

$$T_{cusum,t} = c_{cusum} \cdot \sigma_{e,t} \ , \quad (6)$$

where $c_{cusum}$ is a constant. A measurement is flagged anomalous every time $T_{cusum,t}$ has been exceeded. To improve the precision of anomaly end time detection, we use an upper bound on $S_t$ to let $S_t$ decrease faster after $x_t$ has decreased.

The presented algorithm relies only on the forecasted value and the measured value. Due to the daily periodicity of network traffic and the quick rises and falls of network utilization during mornings and evenings, respectively, the detection algorithm may benefit from a longer history. Our second considered detection algorithm uses the Holt-Winters Additive forecasting method for modeling seasonal components [19]: By adding a linear and a seasonal component to a base signal, the next value is forecasted. As a consequence of the daily periodicity of network traffic, we use day-long seasons. Since we do not identify a significant linear trend in network traffic at this timescale, we disregard the linear component. The weighted mean (EWMA) of the base component is calculated based on previous measurements on the same day, while the

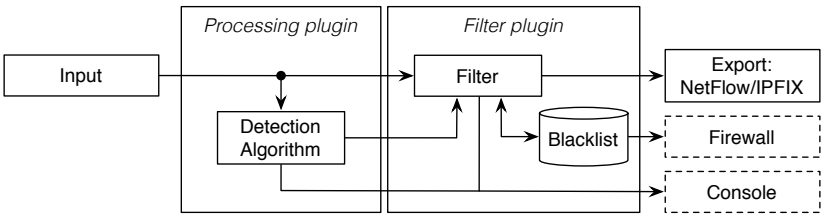



Fig. 4: Prototype Architecture.

mean of the seasonal component is calculated over values at the same time in previous days. We define these two components as follows:

$$b_t = \alpha \cdot (x_t - s_{t-m}) + (1-\alpha) \cdot b_{t-1} \quad (7)$$

$$s_t = \gamma \cdot (x_t - b_t) + (1-\gamma) \cdot s_{t-m} \quad (8)$$

$$\hat{x}_{t+1} = b_t + s_t \ , \quad (9)$$

where $b_t$ and $s_t$ are the *base* and *seasonal* components of the forecasted value $\hat{x}_{t+1}$, respectively, $m$ is the season length (*i.e.* number of measurement intervals per day, since network traffic shows daily periodicity), and $\gamma \in (0,1)$ a parameter which determines the rate in which previous values are discarded. The previous values in this case are not from the previous measurement interval, but from the same interval in the previous season (*i.e.* day). The initial base value is set to the average of all measurement values in the first season. Therefore, a training period of one season is needed.

The use of day-long seasons and small measurement intervals results in a large number of measurement values per season. To support our requirement of being lightweight, we only store seasonal values every hour and interpolate between those. This also reduces measurement noise, which would otherwise imprint in the seasonal values. Besides that, precautions need to be taken to not let measurements during anomalies influence the forecasts, which can be accomplished by not updating $s_t$, $b_t$ and $e_t$ during an anomaly. This is because we aim at forecasting non-anomalous network behavior. Another improvement made to the algorithm is to separate algorithm states for weekdays and weekends, since the traffic behavior usually varies significantly between these types of days. As such, forecasting of weekend days is done based on the traffic behavior of the previous weekend, instead of working days. Analogously for weekdays.

Siris *et al.* have shown that an interval between 5 and 20 seconds yields best results for detecting flooding attacks using the CUSUM method [15]. Our measurements have shown that an interval of 5 seconds indeed results in the most accurate detection results, but an extensive comparison of various interval lengths is out of the scope of this paper.

## IV. Validation Setup

This section describes the setup used for validating our work. We start by discussing the developed prototype in Section IV-A, after which we provide details on the two datasets used for validating our requirements in Section IV-B.

### A. Prototype

Our prototype implements both the traffic measurements based on the metric chosen in Section II (*i.e.* number of flow records creations) and the detection algorithms presented in Section III. It has been developed as a plugin for INVEA-TECH's FlowMon platform, which has been selected both because we have full control over it in our networks, and because of its highly customizable architecture based on plugins for data input, flow record processing, filtering, and export. The prototype has been designed as a hybrid processing and filtering plugin. Default input and export plugins from INVEA-TECH are used for packet capturing and NetFlow and IPFIX data export. Information gathered by the prototype, such as detected anomalies, are sent to a console. The complete architecture is depicted in Fig. 4.

The intrusion detection module has been implemented as a processing plugin. After every measurement interval, the algorithm is run and the measurement sample is classified as benign or malicious. This result is then passed on to a filter plugin, which is used for attack mitigation. When measurement samples are classified as benign, the corresponding flow records are passed on to the export plugin. Otherwise, the filter plugin identifies attackers as soon as an attack has been detected. Attackers are identified by counting the number of exported flow records per source IP address. When more than $F$ flow records per second with less than 3 packets and identical source IP address have been exported, the address is added to a *blacklist*. Measurements have shown that $F = 200$ is high enough to ascertain that a blacklisted host was flooding a network or host, and that benign hosts should never become blacklisted. In the case of attacks with spoofed IP addresses, one could also consider blacklisting destination addresses. We have measured the effects of this approach as well, but source address blacklisting has yielded slightly better results.

After identification of the attackers, the filter plugin performs two actions, which correspond to the contributions identified in Section I:

1) Firewall rules are composed and sent to a firewall to block the attackers' traffic. This corresponds to $(1)$ in Fig. 1.
2) Flow records with the attackers' IP addresses are filtered to reduce the stream of flow records sent to the collector. This corresponds to $(2)$ in Fig. 1.

When an anomaly has ended, the composed rule is removed from the firewall, counting of exported flow records is stopped,

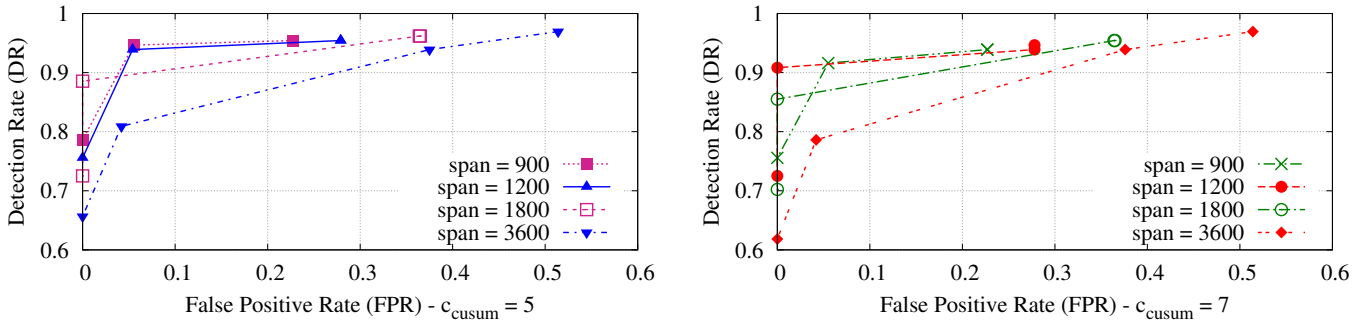



Fig. 5: Receiver Operating Characteristics (ROC) for Algorithm 1.

TABLE I: Datasets

| | Dataset 1 | Dataset 2 |
|---|---|---|
| **Duration** | 14 days | 10 days |
| **Period** | August/September 2012 | October/November 2012 |
| **Flows** | 10.0 G (717.0 M per day) | 6.7 G (668.9 M per day) |
| **Packets** | 257.1 G (18.4 G per day) | 186.7 G (18.7 G per day) |
| **Bytes** | 134.5 T (9.6 T per day) | 128.5 T (12.8 T per day) |
| **Anomalies** | 131 | 11 |
| **Anomaly duration** | Minimum: 5s<br>Average: 5m, 55s<br>Maximum: 2h, 41m, 50s | Minimum: 5s<br>Average: 15m, 55s<br>Maximum: 2h, 48m, 55s |

and all counters are reset. The filtering of flow records is stopped after $T_{idle}$ seconds, where $T_{idle}$ is the idle timeout of the flow exporter, to make sure that flow records in the exporter's *flow cache* that still belong to the attack are filtered.

### B. Dataset

The dataset used for validating the detection algorithms has been captured on a backbone link of the CESNET network in August/September 2012. This link has a wire-speed of 10 Gbps with an average throughput of 3 Gbps during working hours. The dataset comprises 14 days of measurements, composed of the number of flow record creations, packets and bytes per measurement interval (details are listed in Table I, Dataset 1). To establish a ground truth for validation, we have manually identified anomalies that show a high intensity in the number of flow records. Samples belonging to an anomalous interval are labeled malicious. Other samples are labeled benign.

## V. Validation Results

In this section we validate whether our approach meets the requirements identified in Section I. We start by validating the accuracy in Section V-A, mainly because of the fact that an intrusion detection module with a poor accuracy would be useless in any setup. After choosing the algorithm that performs best in terms of accuracy, we validate the response time and performance footprint of this algorithm in Section V-B and Section V-C, respectively.

### A. Accuracy

The accuracy of both detection algorithms is visualized by the Receiver Operating Characteristics (ROC) curves shown in Fig. 5 and 6. The curves show the impact of the constant $c_{threshold}$ on the Detection Rate (DR) and the False Positive Rate (FPR). The DR is a measure for the number of attacks that have been detected correctly and is defined as follows [20]:

$$DR = \frac{\#\{\text{detected attacks}\}}{\#\{\text{attacks}\}} \tag{10}$$

The total number of attacks is determined by considering consecutive malicious samples to belong to the same attack. An anomaly is considered detected if approximately 50% of the samples is flagged malicious. The FPR is the ratio between the number of samples incorrectly flagged malicious and the number of samples labeled benign. In contrast to the more common practice of plotting the True Positive Rate (TPR, ratio between the number of samples correctly flagged malicious and the number of samples labeled malicious) versus the FPR, we plot the DR versus the FPR. This is because we do not require our algorithm to flag *all* samples of an anomaly, as long as the ones with a high intensity are catched. Each curve in the plots shows the accuracy for a different combination of *span* and $c_{cusum}$. *Span* represents the length in seconds of the history considered by the detection algorithm. As the algorithm is only aware of the number of measurement intervals and not of durations, we convert this time window to measurement intervals by dividing it by the length of a measurement interval. As such, it is used for calculating $\alpha$ ($\alpha = \frac{2}{N+1}$, where $N$ is the number of intervals [16]) and for determining the number of values considered in calculating the standard deviation of forecasting errors $\sigma_{e,t}$. Besides *span* and $c_{cusum}$, all other parameters have been fixed: $M_{min} = 7000$ (Algorithm 1 and 2), $\gamma = 0.4$ (Algorithm 2).

Several observations can be made regarding the performance of Algorithm 1 in Fig. 5. First, it is clear that the difference in $c_{cusum}$ has little impact on the DR and FPR. Each pair of curves with the same *span* shows very similar growth. Second, increasing the *span* has little impact on the DR as well, but it increases the FPR significantly. This is because

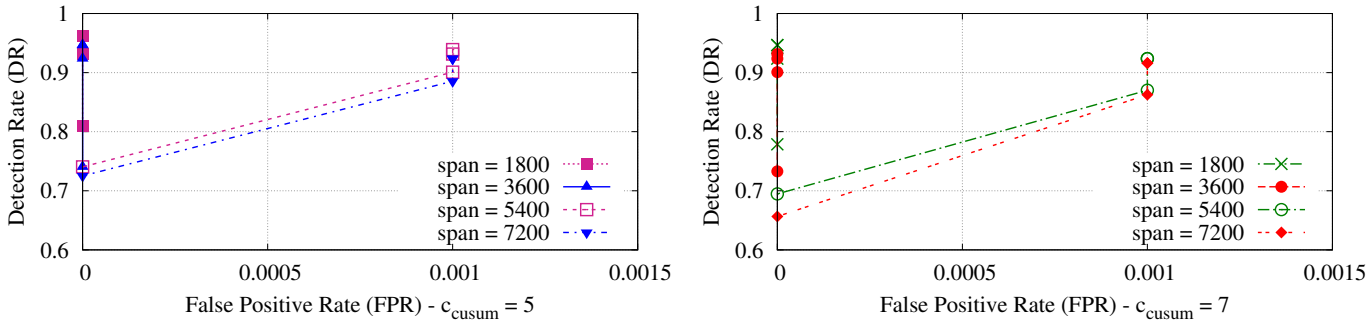
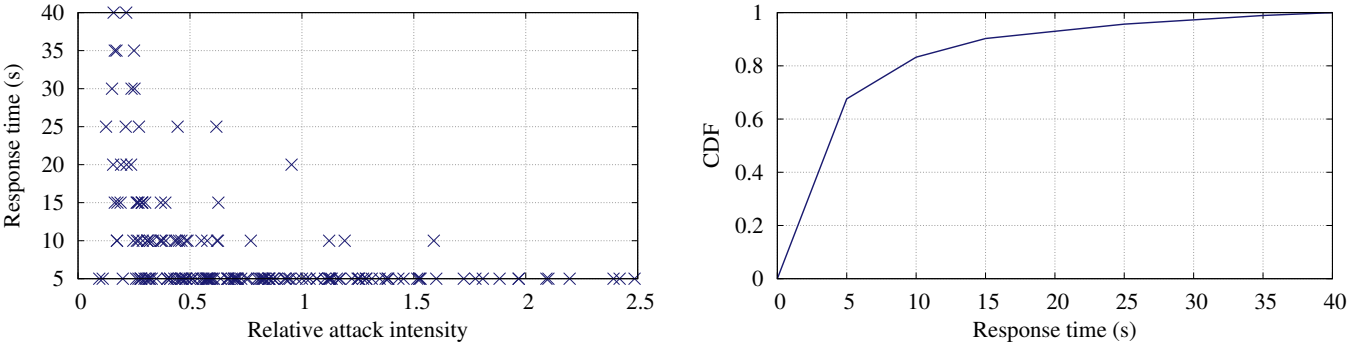



Fig. 6: Receiver Operating Characteristics (ROC) for Algorithm 2.




(a) Response times for various attack intensities.

(b) CDF of the response time.

Fig. 7: Response Times of Algorithm 2.

the forecast adapts slower to network traffic changes, such as diurnal patterns, and small deviations in the measurements are (incorrectly) flagged as malicious. Third, increasing $c_{threshold}$ affects the DR negatively: The highest DRs in the figure are achieved when the lowest $c_{threshold}$ is used. This is because the resulting higher upper threshold $T_{upper,t}$ will cause certain anomalies to stay below the threshold, resulting in a higher number of FNs. In the case of Fig. 5, $c_{threshold} \in \{1.5, 2, 3, 5\}$. In our experiments, a *span* of 900 seconds and a $c_{threshold}$ of 3 yield the most optimal combination of a high DR (92%), while maintaining a low FPR (6%). In a typical deployment scenario as shown in Fig. 1, however, this FPR is unacceptable as benign hosts may be blocked erroneously by our approach.

The ROC curve for various combinations of parameter values for Algorithm 2 is shown in Fig. 6. Similar parameter values as for Algorithm 1 yield similar DRs, while the FPR is significantly lower, namely between 0% and 0.01%. Higher values of $c_{threshold}$ yield lower DRs, for the same reason as described for Algorithm 1. Again, $c_{threshold} \in \{1.5, 2, 3, 5\}$. When a *span* $> 3600$ seconds is used, the FPR increases slightly for small values of $c_{threshold}$, although still being very small (0.1%).

In general, we conclude that Algorithm 2 is more suitable as a detection algorithm in our situation than Algorithm 1, since the FPRs are much lower while similar (high) DRs are maintained. We therefore conclude that the higher accuracy of this algorithm should excuse the additional performance footprint on the flow exporter. In the remainder of this section, we will therefore only consider Algorithm 2.

### B. Response Time

The main goal of this paper is to perform flow-based intrusion detection in near real-time. An important metric in the validation is therefore the response time. We define the response time as the time between the moment in which the algorithm detects an anomaly and the beginning of the anomaly. A scatter plot showing response times for various attack intensities is shown in Fig. 7a, where we define the relative attack intensity as the fraction between the forecasting error (see (3)) and the forecasted number of flow records:

$$\frac{e_t}{\hat{x}_{t+1}} \tag{11}$$

The response time is always a multiple of 5 seconds, as this is the length of our measurement intervals. A response time of 5 seconds means that an anomaly has been detected within the same sample as the anomaly has started. As shown in the figure, most anomalies with a relative intensity larger than 0.3 are detected within 10 seconds. Outliers are the result of attacks that do not reach their full intensity right from the start. Anomalies with a relative intensity $< 0.3$ are mostly detected within 40 seconds. However, these anomalies are not the main

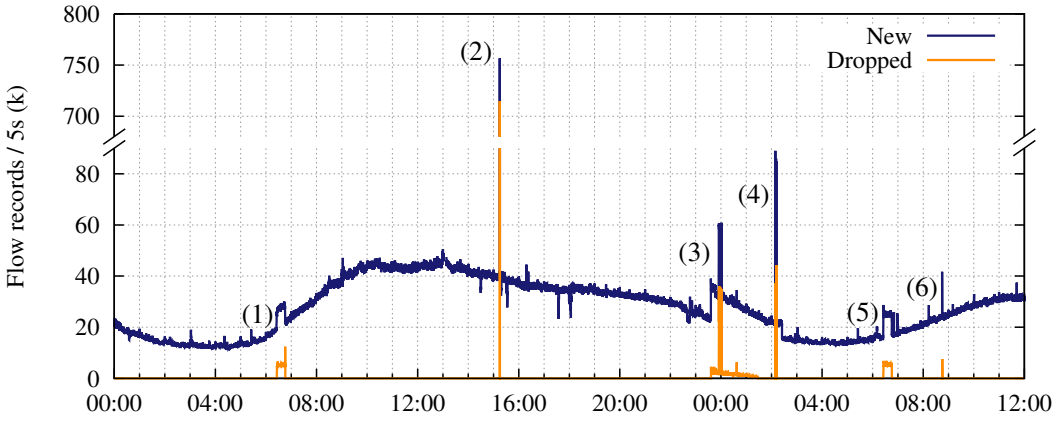



Fig. 8: Prototype Mitigation Results.

target of our work as their potential damage to networks and hosts will be limited. Another view on the response times of the algorithm is shown in Fig. 7b, where the CDF is plotted for each potential response time. It can be observed that 68% of all anomalies is detected within 5 seconds and 90% of the anomalies within 15 seconds. Note that these response times are even lower than typical idle timeouts of flow exporters, as shown in Section I.

An example of the prototype in operation is shown in Fig. 8. The figure shows the number of flow record creations, as measured by the processing plugin per measurement interval of 5 seconds, and the number of flow records dropped by the filter plugin per measurement interval, over a period of 36 hours. This measurement period is a subset of Dataset 2 (see Table I). Several large anomalies can be identified, labeled as (1)-(6). The anomalies (1), (5) and (6) are clearly smaller than the others and are dropped largely or completely by the filter plugin. However, the main focus of our work is on very large anomalies, such as the anomalies marked as (2), (3) and (4). Anomaly (2) consists of 755k flow records per 5 seconds, while roughly 40k flow records have been forecasted. Out of these 755k flow records, our prototype is able to mitigate 715k. Anomaly (3) and (4) are both part of one longer anomaly, which is dropped partly throughout the duration of the attack. Anomaly (3) is mitigated completely, as the number of passed flow records roughly equals the forecasted number of flow records for measurement intervals during the attack (23k). Anomaly (4) is mitigated partially, where about 50% of the total number of flow records is dropped. When anomalies have not been dropped completely, one or more attackers generated less flows than the threshold of $F = 200$ per second. We do not consider this problematic since the number of passed flow records (40k per measurement interval) is in principle not causing collector overload, as this number equals the number of benign flow records at midday. Anomalies outside the visualized part of Dataset 2 (*i.e.* anomalies that are not shown in Fig. 8) have been detected and mitigated similarly to the presented anomalies.

### C. Performance Footprint

The last identified requirement for our intrusion detection module is that it is lightweight, as the operation of an exporter is considered time-critical. To verify whether our prototype fulfills this requirement, we have run the exporter process on our flow exporter both with and without our prototype. On average, the exporter process consumes less than 5% more CPU time when the processing and filter plugins are loaded. The memory footprint of the plugins depends on the size of an attack, as the number of flow records per IP address are counted under such circumstances. Our measurements have shown that the plugins never consume more than 20 MB of memory when the network is under attack.

## VI. Feasibility

In this paper we have used INVEA-TECH's FlowMon platform for validating all identified requirements for our intrusion detection module. This dedicated flow export platform has been chosen both because we have full control over it in our networks and because of its highly customizable architecture. An alternative approach would have been to use a high-end forwarding device with flow export capabilities.

Several platforms could have been chosen for implementing this, as long as they support some form of scripting for implementing the algorithm and retrieving statistics from the flow cache of the exporter. One option is Cisco's IOS, which supports scripting based on the Tool Command Language (TCL) as of version 12.3(2)T and 12.2(25)S. This provides administrators both a means to automate CLI command sequences and perform processing on information gathered from CLI commands and SNMP MIBs. Another option is Juniper's Junos, which is a specific command shell on top of a BSD-based kernel. It provides a full UNIX-level shell to administrators. Normal UNIX commands can therefore be used, which makes it straightforward to run customizations on a high-end device.

Although our approach could already be implemented for these platforms, performing intrusion detection based on our

approach may be rather intrusive to a forwarding device. Before deployment in a production network, its behavior under load needs to be monitored carefully in a testing environment. This is because the scripts are processed in software by a generic CPU, which might become the bottleneck of the system as soon as the system load increases.

## VII. Conclusions

This paper has presented an extension for NetFlow and IPFIX flow exporters that detects and mitigates malicious network traffic in a near real-time fashion. Our results show that we can detect flooding attacks within seconds, effectively overcoming the detection delays incurred in traditional flow-based monitoring systems. In addition, we can instruct other security systems (*e.g.* firewalls) to block malicious traffic, and filter flow data of malicious traffic to prevent flow collector overload. Deployment of a prototype in the CESNET backbone network has proved both the applicability and successful operation of our approach.

Validation of our work has shown that response times as short as 5 seconds can be achieved, which easily fulfills our design target of reducing detection delays to at least 10%. This is the smallest possible value for our prototype, as it is based on measurement intervals of 5 seconds. On the one hand, related work has shown that smaller measurement intervals result in measurements with too much noise, reducing the accuracy of anomaly detection algorithms. On the other hand, larger measurement intervals result in higher detection delays and again reduced accuracy. Another requirement for our intrusion detection module has been a limited performance footprint, which has been targeted to be no more than 10%. Our prototype has shown never to consume more than 5% additional CPU time and that the additional memory usage is negligible. Finally, our prototype has a high detection rate, while maintaining a very low number of false positives. This shows that our approach is accurate.

As future work, we plan to implement this work on high-end forwarding devices, such as switches and routers. Our feasibility study has shown that this is possible, although practical implications have to be studied. In addition, we consider specifying and implementing a *meta-flow record*. When malicious flow data is filtered before it arrives at a flow collector, this special flow record should fill the gap of missing data. It should contain aggregated flow data, such as the number of flow records, packets and octets that have been filtered out by the flow exporter.

## Acknowledgements

This paper has been supported by the EU FP7-257513 UniverSelf Collaborative Project, SURFnet's GigaPort3 project for Next-Generation Networks, the EU FP7 Flamingo Network of Excellence project (ICT-318488), the research programme MSM 0021630528, the IT4Innovations Centre of Excellence CZ.1.05/1.1.00/02.0070 and the grant BUT FIT-S-11-1.

## References

[1] The Spamhaus Project, "Spamhaus," 2013, accessed on 9 August 2013. [Online]. Available: http://www.spamhaus.org

[2] The New York Times, "Firm Is Accused of Sending Spam, and Fight Jams Internet," March 2013, accessed on 9 August 2013. [Online]. Available: http://www.nytimes.com/2013/03/27/technology/internet/online-dispute-becomes-internet-snarling-attack.html

[3] Ars Technica, "Can a DDoS break the Internet? Sure just not all of it," April 2013, accessed on 9 August 2013. [Online]. Available: http://arstechnica.com/security/2013/04/can-a-ddos-break-the-internet-sure-just-not-all-of-it/

[4] Arbor Networks, "DDoS and Security Reports: The Arbor Networks Security Blog," December 2012, accessed on 9 August 2013. [Online]. Available: http://ddos.arbornetworks.com/2012/12/lessons-learned-from-the-u-s-financial-services-ddos-attacks/

[5] B. Claise, "Cisco Systems NetFlow Services Export Version 9," RFC 3954 (Informational), Internet Engineering Task Force, October 2004. [Online]. Available: http://www.ietf.org/rfc/rfc3954.txt

[6] G. Sadasivan, N. Brownlee, B. Claise, and J. Quittek, "Architecture for IP Flow Information Export," RFC 5470 (Informational), Internet Engineering Task Force, March 2009. [Online]. Available: http://www.ietf.org/rfc/rfc5470.txt

[7] B. Claise, S. Bryant, S. Leinen, T. Dietz, and B. Trammell, "Specification of the IP Flow Information Export protocol," RFC 5101 (Standards Track), Internet Engineering Task Force, January 2008. [Online]. Available: http://www.ietf.org/rfc/rfc5101.txt

[8] A. A. Galtsev and A. M. Sukhov, "Network Attack Detection at Flow Level," in *Proceedings of the 11th International Conference and 4th International Conference on Smart Spaces and Next Nenera-tion Wired/Wireless Networking, NEW2AN'11/ruSMART'11*, ser. Lecture Notes in Computer Science, 2011, pp. 326–334.

[9] M.-S. Kim, H.-J. Kang, S.-C. Hong, S.-H. Chung, and J. W. Hong, "A Flow-based Method for Abnormal Network Traffic Detection," in *IEEE Network Operations and Management Symposium, NOMS 2004*, vol. 1, April 2004, pp. 599–612.

[10] G. Münz and G. Carle, "Real-time Analysis of Flow Data for Network Attack Detection," in *10th IFIP/IEEE International Symposium on Integrated Network Management, IM 2007*, 2007, pp. 100–108.

[11] R. Hofstede and A. Pras, "Real-Time and Resilient Intrusion Detection: A Flow-Based Approach," in *Dependable Networks and Services. Proceedings of the 6th International Conference on Autonomous Infrastructure, Management and Security, AIMS 2012*, June 2012, pp. 109–112.

[12] R. Sadre, A. Sperotto, and A. Pras, "The Effects of DDoS Attacks on Flow Monitoring Applications," in *IEEE Network Operations and Management Symposium, NOMS 2012*, 2012, pp. 269–277.

[13] Cisco Systems, "Introduction to Cisco IOS NetFlow - A Technical Overview," White paper, May 2012.

[14] Juniper Networks, Inc., "flow-inactive-timeout – Technical Documentation," November 2012, accessed on 9 August 2013. [Online]. Available: http://www.juniper.net/techpubs/en_US/junos/topics/reference/configuration-statement/flow-inactive-timeout-edit-forwarding-options.html

[15] V. A. Siris and F. Papagalou, "Application of Anomaly Detection Algorithms for Detecting SYN Flooding Attacks," *Computer Communications*, vol. 29, no. 9, pp. 1433–1442, 2006.

[16] N. Ye, C. Borror, and Y. Zhang, "EWMA Techniques for Computer Intrusion Detection Through Anomalous Changes in Event Intensity," *Quality and Reliability Engineering International*, vol. 18, no. 6, pp. 443–451, 2002.

[17] H. Wang, D. Zhang, and K. Shin, "Detecting SYN flooding attacks," in *INFOCOM 2002. Twenty-First Annual Joint Conference of the IEEE Computer and Communications Societies*, vol. 3, 2002, pp. 1530–1539.

[18] G. Münz and G. Carle, "Application of forecasting techniques and control charts for traffic anomaly detection," in *Proceedings of the 19th ITC Specialist Seminar on Network Usage and Traffic*, 2008.

[19] J. D. Brutlag, "Aberrant Behavior Detection in Time Series for Network Monitoring," in *Proceedings of the 14th Conference on Systems Administration, LISA 2000*, 2000, pp. 139–146.

[20] A. Sperotto, M. Mandjes, R. Sadre, P. T. de Boer, and A. Pras, "Autonomic Parameter Tuning of Anomaly-Based IDSs: an SSH Case Study," in *IEEE Transactions on Network and Service Management*, vol. 9, no. 2, 2012, pp. 128–141.