An Efficient Online Anomalies Detection Mechanism for High-speed Networks

Osman Salem, Sandrine Vaton, Annie Gravey
ENST Bretagne
Department of Computer Science
Brest, France
Email: {osman.salem, sandrine.vaton, annie gravey}@enst-bretagne.fr

***Abstract*—In this paper, we propose an efficient framework for online detection and identification of network anomalies, in early stage of its occurrence, to quickly react by taking the appropriate countermeasures. The proposed framework is based on online detection of change point in a multi-layer reversible sketch, which aggregates multiple data streams from high speeds links in a stretched database. To detect network anomalies, we apply non-parametric multi-channel CUSUM algorithm at the counter value in each bucket of the proposed reversible sketch, in order to undermine flows with abrupt change, and to discover the keys of culprit flows via sketch inversion. Theoretical framework for detection and classification of attacks are presented. We also give the results of our experiments analysis at two data traces collected with Netflow and Endace DAG-3 card. Our analysis results from real-time internet traffic and online implementation show that our proposed architecture is able to detect culprit flows quickly with a high level of accuracy.**

I. Introduction

Security threats for computer network have risen significantly, which include viruses, worm-based attacks, port scanning, denial of service (DoS), distributed DoS (DDoS), etc. As the use of internet in society become crucial, it will attract more intentional attacks. While many anomaly detection mechanisms have been introduced to undermine attacks, the effectiveness of these models is largely dependent of traffic distributions parameters. They lack the capability of handling shape irregularities and unpredictable large fluctuations of real IP traffic.

Furthermore, most existing intrusion detection systems (IDS) reside at end host or end router. They usually lack scalability in handling large state space traffic information at high speed links (backbone links), where even the handling at flow level is very costly in terms of per-flow state storage requirement, and update/search operations complexity. Flows are usually characterized by 5 tuples (e.g. Netflow) : source and destination IP address, source and destination port, and protocol number. This means monitoring flows state space requires updating and handling a database table of size 2^{104}.

In this paper, we consider the problem of online detection and identification of network anomalies at high speed links, in order to cope with attacks as soon as possible, by taking countermeasure actions. Useful information about victims and attackers are provided, and can be exploited to shield the victim server. The embedded information in alert message depends on the type of attack.

A naïve idea for monitoring flows at high speed link, is to maintain a database for active flows for a fixed time interval T, and to query database for heavy hitter (or massive) flows. However, this strategy is not scalable, where spatial and temporal complexities for update and query operations prevent its use for handling a large number of flows at high speed links in real time manner.

In response to these limitations, a special data structure based on $k - ary$ hash tables (Fig. 1), called sketch [1]–[3], was proposed and used to handle large state space, with a small memory requirement and linear computational (update/query) complexity. It is a multi-stage bloom filter based on random aggregation, where flows identifier (key) are hashed to index into a set of buckets (or cells) in different stages using k different hash functions. These hash functions are generally chosen to reduce collision effect. Sketch has proven to be useful in many data stream computation applications [4], especially in detecting hierarchical heavy hitters [5], [6] and heavy changes [7] in network monitoring.

To use sketch in context of network anomalies detection, IP flows are typically classified by some fields combination in packet header, such as destination IP address (DIP), or source and destination IP address (SIP/DIP), etc. This flow identifier is a key used to update k^{th} hash table by its associated value $(key, value)$. The value is a reward associated with key, and which can be the number of: packets, bytes, connection request ($\#SYN$), number of half open connection ($\#SYN - \#SYNACK$), or other flow characteristics.

Recent work with Count-Min Sketch (*CMS* [4]) has shown that random aggregation of flow does not significantly disrupt flow variations. The *CMS* algorithm can indicate if a given key exhibits large value, and even one can query sketch about occurrence frequency for a given key, i.e. the accumulated value with a given key.

However, sketch is based on universal hash functions, which are not reversible. Consequently, we cannot use sketch to report the set of heavy hitter flows, because sketch does store any information about its key entry. Thus, the only way to get all heavy keys (which exhibit heavy values) is to test all entries, in order to determine those mapped to heavy buckets, which mean when monitoring high speed links, all keys must be recorded and verified. Unfortunately, this approach is neither scalable nor efficient.

Even if one can assume that we will stay far from worst case, and only a relative small number of keys ($SIP|DIP$) with respect to universe of size 2^{64}, will appear during a T time interval, still the question of attack detection, where a lot of approaches have tackled this problem by verifying if the values of the buckets associated to a given key are heavy hitter or not. Recent work in [8] lookup for heavy buckets in the sketch resulted from the difference between current epoch and time series forecasting sketches. However, focusing at heavy hitter is not very useful in practice, due to large traffic fluctuations and variations, and heavy hitter results do not necessarily correspond to malicious flows.

Our proposed framework for network anomalies detection is carried out by applying sequential anomalies detection algorithm (Multichannel Non-Parametric CUSUM MNP-CUSUM [9], [10]) over the buckets of sketch data structure. An additional multi-layer reversible sketch (*MLRS*) is used for software efficient sketch inversion, in order to pinpoint culprit flows after the detection of anomalies.

The proposed method begins by recording the packet stream in a compact sketch representation for each T time interval. Afterward, MNP-CUSUM algorithm is used to check the presence of buckets which value deviates significantly from normal behavior. In fact, CUSUM algorithm is able to react quickly when observing abrupt changes in bucket value, while being able to distinguish effect of attack deviation from usual traffic fluctuations. After the detection of anomalies by CUSUM algorithm, we recover associated keys to buckets with raised alarm, by exploiting bucket index in *MLRS*, to pinpoint responsible flows.

In fact, the combination of MNP-CUSUM and sketch improves the efficiency of the detection mechanism, where CUSUM is used to undermine anomalies, and sketch reduces significantly required memory and computational complexity when handling a large amount of data. Proposed method has been analyzed and validated practically. Our results are encouraging in terms of accuracy and response time.

The remainder of this paper is organized as follows. In the following section, we give a brief overview of *CMS* Sketch and MNP-CUSUM mechanisms that are related to our studies. Section III describes our proposed method for detecting change point in a reversible sketch. In section IV, we show measurement based verification and evaluation of the effectiveness of our proposed framework. Finally, section V presents concluding remarks and the future work.

II. Background

In this section, we briefly survey the underlying *CMS* sketch data structure and MNP-CUSUM theory related to our work.

A. Stretching data stream in sketch

Let $S = s_1 s_2 \dots s_n$ be the set of input stream that arrives sequentially, item by item [4]. Each item $s_i = (\kappa_i, v_i)$ is identified by a key $\kappa_i \in U$ drawn from a fixed universe of items U. A reward (or frequency occurrence) value $v_i \in \mathbb{R}$ is associated with each key. The arrival of item with key κ_i increments its associated counter in the j^{th} hash table by ν_i ($C_{j,h_j(\kappa_i)} += \nu_i$), as shown in Fig. 1. The update procedure is realized by d different hash function, chosen from the set of 2-universal hash function $H_i = \{((a_i x + b_i) \bmod P) \bmod w\}$, to uniformly distribute κ_i over hash tables and to reduce collision.

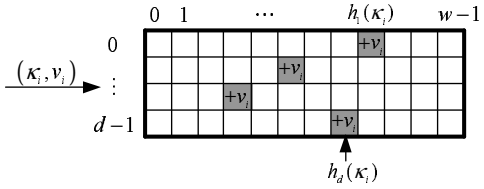


Fig. 1. Sketch data structure.

The Count-Min point query returns an estimate of the counter for a given key, as the minimum of d counters value ($\hat{s}_k = \min_{0 \le j < d} \{C[j][h_j(\kappa_i)]\}$).

B. MNP-CUSUM change detection algorithm

In contrast to the most widely used techniques for anomalies detection (heavy hitter), sketch can be used with various sophisticated sequential detection procedures to uncover anomalies. In this paper, we will focus at sequential MNP-CUSUM [9], [10] algorithm, due to its high precision, low computational overhead and modest storage requirements.

CUSUM relies on two phases: training and detection. In training phase, it establishes and updates a dynamic behavior profiles for normal flows, and in second phase, it uses log likelihood ratio to detect any kind of abrupt deviation from well established flow profile, through using the assumption that most anomalies induce a change in distributions of monitored parameters.

Let $\{X_{i,j}^{nT}, 1 \le i \le d, 1 \le j \le w\}$ be the value of each bucket during the n^{th} time interval. Observations $X_{i,j}^{nT}$ are i.i.d with a *pdf* $f_{ij,\gamma_0}(x)$ for $n < t_a$ (before attack occurrence) and with another *pdf* $f_{ij,\gamma_1}(x)$ (after attack), where t_a is the instant of attack detection. M-CUSUM test statistical hypotheses H_{ij} (eq. 1) to detect abrupt change in bucket with index (i, j) at the time epoch $n = t_a$:

$$H_{ij,0}: \quad \gamma_{ij} = \gamma_0 \qquad \text{versus} \qquad H_{ij,1}: \quad \gamma_{ij} = \gamma_1 \quad (1)$$

Where γ_0 and γ_1 are respectively the *pdf* parameters before and after change occurrence. The detection of anomaly is given by testing M-CUSUM function value G_{ij}^{nT} (if $G_{ij}^{nT} > threshold\ h$ then raises alarm), given by eq. 2:

$$G_{ij}^{nT} = \left\{ 0, G_{ij}^{(n-1)T} + \ln\left(\frac{P(X_{ij}^{nT}|\gamma_1)}{P(X_{ij}^{nT}|\gamma_0)}\right) \right\}^{+} \wedge G_{ij}^{0} = 0 \quad (2)$$

However, due to large variation in traffic pattern, and lack of consensus on network traffic characteristics (self similar, exponential inter-arrival, heavy tailed length, etc.), we consider non *i.i.d* traffic characteristics, and we will apply Multichannel Non-Parametric adaptive CUSUM (MNP-CUSUM [11]) to every sketch bucket $B_{i,j}$, as a detection mechanism insensitive

to traffic patterns. The test function of MNP-CUSUM is updated using the following formula:

$$G_{ij}^{nT} = \left\{ G_{ij}^{(n-1)T} + w_{ij}(X_{ij}^{nT} - \mu_{ij} - \varepsilon\hat{\theta}_{ij}) \right\}^{+} \wedge G_{ij}^{0} = 0 \quad (3)$$

Where $\{y\}^{+} = \max(0, y)$, μ_{ij} and θ_{ij} are the pre-change and post-change mean value of the corresponding bucket. Eq. 3 means monitoring the sequential value of bucket to detect change in the mean value in unknown instant. The value of μ_{ij} can be estimated in a self learning phase and updated periodically using EWMA (Exponential Weighted Moving Average) formula given in eq. 4. Hence μ_{ij} is supposed to be known.

$$\mu_{ij}^{nT} = \alpha\mu_{ij}^{(n-1)T} + (1-\alpha)X_{ij}^{nT} \quad (4)$$

The value of θ_{ij} are unknown and should be dynamically estimated online, subject to condition that $\theta_{ij} > \mu_{ij}$, and attack leads to change in the mean value of $\mu_{ij} \rightarrow \theta_{ij}$. $\hat{\theta}_{ij}$ is an estimation of the unknown mean θ_{ij} and depends on the past observations, w_{ij} is a positive weight for performance adjustment, and usually set to 1. ε is a tuning parameter chosen from $[0, 1]$. We refer to [11] for a complete reference about these parameters.

III. Proposed approach

Our proposed framework is based on 2 data summary architecture: a Multi-Layer Reversible Sketch (*MLRS*) and a Count-Min Sketch (*CMS*) as shown in Fig. 2. Operations of the proposed framework are performed by two steps. First, it continuously updates the two sketches (*MLRS* and *CMS*) counters from input data stream (κ_i, v_i) for a fixed time interval T. Secondly it applies MNP-CUSUM in the background at each bucket to detect anomalies. Afterward we identify and output keys that mapped to buckets with a CUSUM triggered alarm.

However, reversing sketch is a difficult operation, where there is only two existing approaches in the literature. The first [12] is based on intuitive idea by storing all keys in T time interval, and achieve verification by re-hashing (hashing for the second time) the set of stored keys at the end of each interval. This strategy is inefficient in term of storage space and update speed for the list of keys.

The second approach [8], [13] is based on modular hashing and mangling via Galois Field $GF(2^n)$ operators, used in Reed-solomon coding technique. However, GF is based on binary polynomial representation and bit by bit operations. This strategy is very complex and more efficient for hardware implementation, as it was done in [8].

Our idea to reverse sketch is based on exploiting index in an additional multi-layer sketch (Fig. 2), where indexes are used to store keys. In fact, the multi-layer sketch is used in the same way of *CMS* sketch, where the arrival of each key increments its counter. However, each key has l counter (one by layer), where we split the key of N bit into $l \times w$ bit, with $w_0 = 2^P$, and $l = \lceil N/P \rceil$. P is the number of bits used to split the key, and w_0 is used as layer width in *MLRS*. For

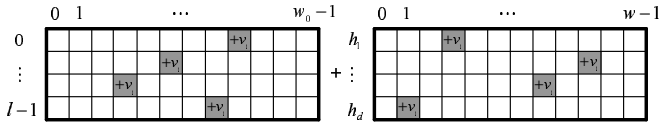


Fig. 2. Multi-Layer Reversible Sketch *MLRS* and *CMS* sketch.

clarification, we assume that sketch key is the DIP address the arrival stream, and the last arrival one is $192.168.41.25$. For *MLRS* with $P = 8$, $w_0 = 2^8$ and $l = 4$, we split DIP into 4-byte, and we increment the counter in each layer ($b[0][192]$, $b[1][168]$, $b[2][41]$, $b[3][25]$) by the associated value v_i.

If we seek to search for victim DIP (or key that maps to buckets with raised alarm by MNP-CUSUM), we can release hierarchical search procedure in *MLRS*. If we don't find at least one bucket with raised alarm in each of the i^{th} $(i \leq l-1)$ first layers of *MLRS*, there is no need to continue searching in other deep layer or through the second *CMS* sketch. Malicious flows must have one alarmed bucket in each layer.

On the other hand, we will begin by the simple case, where we assume that there is at most one alarmed bucket per layer as shown in Fig. 2. To recover key, we concatenate the l binary value of indexes in *MLRS* and we recover the value of suspect key (e.g. DIP). We can not be sure of suspect before verification, where due to collision with other IP prefix, their value becomes large. The suspect key is verified through hashing and verification (by count-min query of CUSUM alarm function) in the *CMS* for confirmation.

In general, even with a different value of width (e.g. 2^{12} or 2^{14}) for the *MLRS*, many buckets in different layers will be subject to collision occurrence, and in some case, we will be found with a bigger set of keys to verify through *CMS* than the original one. Nevertheless, it is important to notify that even if the set of suspect key is larger than departure one, it requires a small memory and fast update time with respect to original list.

To resolve this problem and reduce collision in *MLRS*, we use the idea of IP-mangling presented in [8], but with a software efficient procedures rather than $GF(2^n)$ and its polynomial multiplication and division operations. IP mangling is a reversible procedure, which randomizes the input data in an attempt to destroy correlation between keys, to disperse adjacent keys uniformly at all available buckets. This technique is a bijective function that maps keys in a universe U to U. Each key κ_i is mapped to $y_i = f(\kappa_i)$, with the function f chosen in a way to destroy any correlation between keys, as show in table I. Any bijective function able to destroy correlation between keys, and return a completely random set of keys, can be used. Afterward, we use $f^{-1}(y_i)$ to recover suspect key κ_i from *MLRS*. However, we don't store the set of suspect keys, but once we have a suspect, we realize verification through the *CMS* before integrating it in alert message.

In our experiments, we are interested in mangling keys of size 48 and 64 bit, as we will focus on $DIP|DP$ as key to detect victim server of DDoS/DoS, by associating the number of received SYN as reward, and $DIP|SIP$ to detect either

TABLE I
MANGLING BY OPTIMIZED RC4.

$DIP : DP$	*Mangled value*
81.220.180.191:80	101110001001101001011010$\cdots$0011000111001
81.220.180.192:80	101011100111011101101011$\cdots$1000111111000
81.220.180.193:80	101011110010001110000010$\cdots$0110000001010
$\cdots$	$\cdots$

port scan or source address of attacker (if not spoofed). We use cipher techniques instead of most widely used functions for mangling: $f(\kappa_i) = a.\kappa_i \bmod n$ or $f(\kappa_i) = a \otimes \kappa_i \oplus b$ [8]. Modulo function is not efficient because keys that share the same last k bit, may share the same last k bit after mangling. Galois function is hardware compliant and requires additional memory for fast calculation of polynomial products modulo another prime one. On the other hand, cipher techniques try to uniformly distribute data, and they are robust enough to face statistical cryptanalyst attack. There exist many ciphers encoding techniques that are appropriate for software adaptation.

Most block ciphers use 64-bit blocks or larger. We use the Tiny Encryption algorithm (TEA [14]) for 64-bit key, which based on small number of non linear iterations in Feistel transformation manner (alternation of XOR and ADD). TEA has been proven very powerful for software implementation (few lines of code available in [15]), where it doesn't need any preset table or any additional memory, and its security has been extended with XTEA and XXTEA. We don't rely on its security, just its bijectivity and binary dispersion.

However, we lack the theory to modify block ciphers techniques into 48 bit blocks, and we use an optimized version of RC4 (with code available in [16]), as a stream cipher to mangle each key and hide any correlation between adjacent keys, as shown binary values in table I. These two encoding techniques have been proven to be powerful for mangling in our experimentations in terms of random Hamming distance between adjacent keys.

In Fig. 3, we show the distribution of collision when using direct mapping and mangling in multi-layer sketch update with $P = 12$. Data traces are ~ 5 minutes real bidirectional Internet traffic of 3.10^5 packets, and 42770 flows (here flows are classified by DIP and DP). In fact, used mangling techniques allow to uniformly distributing key over buckets, and prevent collision over IP with same prefix. It is worth noting, that we proved by experiments that the number of collision reduce significantly by increasing the width of multi-layer sketch.

At the end of each time interval T, and after updating counters of the proposed framework continuously in online manner, MNP-CUSUM anomaly detection algorithm run in the background, to update CUSUM function in each bucket, and to raise alarm in bucket where the value of function G_{ij}^{nT} exceeds the threshold. Afterward, we scan *MLRS* for identification of all possible sequence of l bucket (one per layer) with triggered alarm and we realize verification through universal hashing and count-min query over the *CMS*, to ensure that

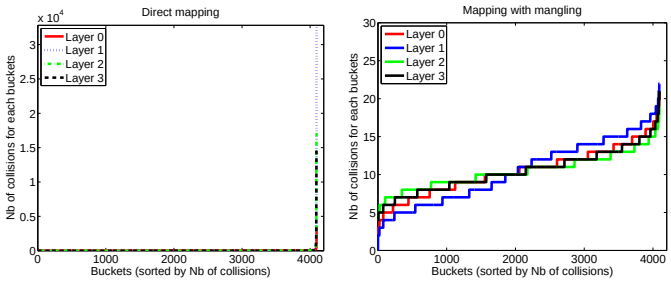


Fig. 3. Distribution of collisions for each bucket.

corresponding buckets with the d universal hash functions have a triggered alarm by CUSUM.

Our proposed framework can be applied to detect different type of attacks, e.g. TCP SYN flood, UDP packet storm, ICMP ping of Death, TCP/UDP PortScan, NetScan, Smurf, etc. Nevertheless, we will only focus at TCP traffic and especially at the number of connection request (SYN). From each arrival packet, we extract three keys ($key_1 = DIP|DP$, $key_2 = SIP|DIP$, $key_3 = SIP|DP$) by the concatenation of binary value of two fields from packet header. These keys are used to update four copies of the proposed framework with the corresponding number of SYN. Our mechanism for attacks classification is given by:

Firstly, we seek to detect victim of DoS/DDoS SYN flooding. We update the counters of first couple of sketches with the key_1 during predefined T time intervals, and we output list L_1 of all victim server $DIP|DP$ as described above. We can also monitor another kind of flooding (ACK, RST, FIN, etc.) by changing the associated reward.

Secondly, the key_2 is used to update a second data structure sketches. Output of this step is list L_2 that contains malicious SIP, which try to scan the ports of given DIP, if this last is not a victim of DDoS/DoS. In contrast, if DIP is in list L_1 (i.e. victim of flooding), we store a list of suspect LoS whose elements are (SIP, DIP, DP), because SIP are suspects of contribution in DDoS/DOS through a static spoofed or not spoofed address.

Thirdly, the key_3 is used for updating the third sketches, where outputs keys are SIP trying to achieve a NetScan activity, if the SIP does not belong to the list LoS. Otherwise, it is the source of DDoS/DoS flooding.

The preceding three steps are used in our implementation to early identify three types of anomaly (DDoS/DoS victim, Scan network and scan port), and provide useful information about victim or attacker. The PortScan and NetScan were chosen for their association with malicious attacks and worms. PortScan are often used by attacker as first step to discover which ports are open on victim machine, to insert malicious code and use it as a zombie. NetScan are usually performed by worms in spreading phase (random scan in code Red, linear in Blaster, bias in code Red II and Nimda, etc.) to gain access to new machines and infect them. Our proposed approach is able to detect all these kinds of scan activities.

IV. Experiments results

In this section, we present performance analysis results of juxtaposing MNP-CUSUM detection algorithm over reversible sketch, for detecting victims of TCP SYN flooding attacks, NetScan and PortScan. We have implemented MNP-CUSUM over sketch in C using the code of CMS available from [17]. We applied the proposed algorithm over many public traces (LBL-TCP-3, Abilene, Auckland, etc.) available from [18], and other unpublished traces used in OSCAR RNRT project (OTIP, ADSL).

In this paper, we present the result of our experiments over 2 set of traces: manually mixed (legal & malicious) traces and OSCAR OTIP trace. To investigate the performance of the proposed framework (true positive and false positive), and to perform a tuning of different parameters, we used 3 hours of Auckland traces (Auckland-VIII 20031202-14h-17h) collected with Endace DAG-3 card, as background traffic to insert various intensity level of anomalies (Fig. 4(c)), due to the lack of public well documented traces with well known attacks. The proposed algorithm does not show any flooding or scanning attack after analyzing the used 3 hours of Auckland traces. Afterward, we modified the timestamp of Auckland traces and we mixed it with only the SYN packets of real distributed attacks. We use the same analysis strategy in [19] for parametric CUSUM, but with real attacks.

Afterward, we present our analysis result over anonymized OTIP traces: 3 days traces collected with Netflow format ($\sim 6.9GB$) and contains a $\sim 896.10^5$ flows.

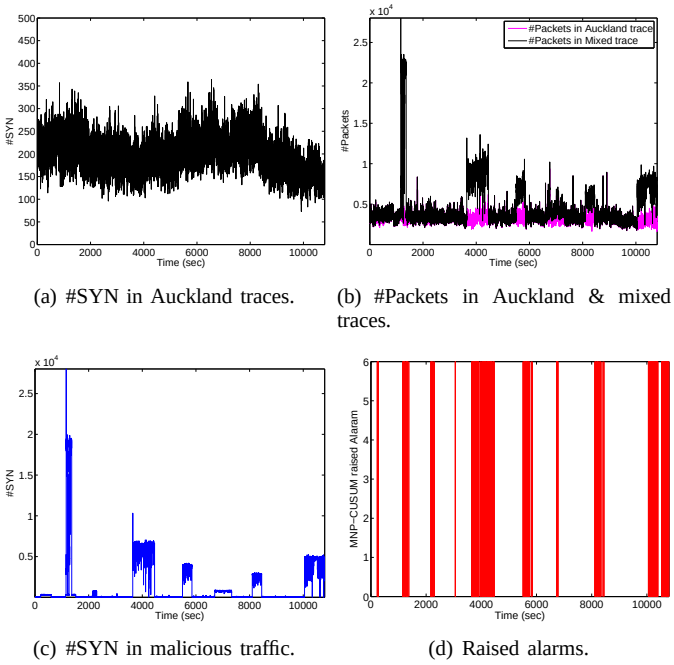
The parameters we considered for the MNP-CUSUM algorithm were: threshold $h = 6$, $\varepsilon = 0.12$, $\alpha = 0.9$. *MLRS* parameters were $P = 14$ unless otherwise noted, $w_0 = 16384$, $l = 4$, $d = 4$ hashing functions from the set of 2-universal hash function, and with the use of tabulation [20]. θ_{ij}^{nT} is updated over some past time window by $\theta_{ij}^{nT} = \delta_{ij}^{nT} + \mu_{ij}^{nT}$, where initial value and adaptive estimation of δ_{ij}^{nT} are detailed in [9].

A. Detection of SYN flooding and scanning attacks

Our first experiment considers detection of different intensity of attack, where the objective was to perform a tuning of different parameters. Fig. 4(a) shows the variation of the number of SYN packets in background traces, Fig. 4(b) presents the total number of mixed packets traces, Fig. 4(c) is the number of malicious SYN traffic and Fig. 4(d) is the raised alarm of MNP-CUSUM algorithm whenever a threshold h is reached. It is worth noting that value of threshold h controls the sensitivity of the attack detection.

In Fig. 5, we present our analysis results over OTIP trace, where we have detected also one NetScan at different time instant by $SIP = 224.87.77.70$ & $DP = 66506$, and non PortScan attack.

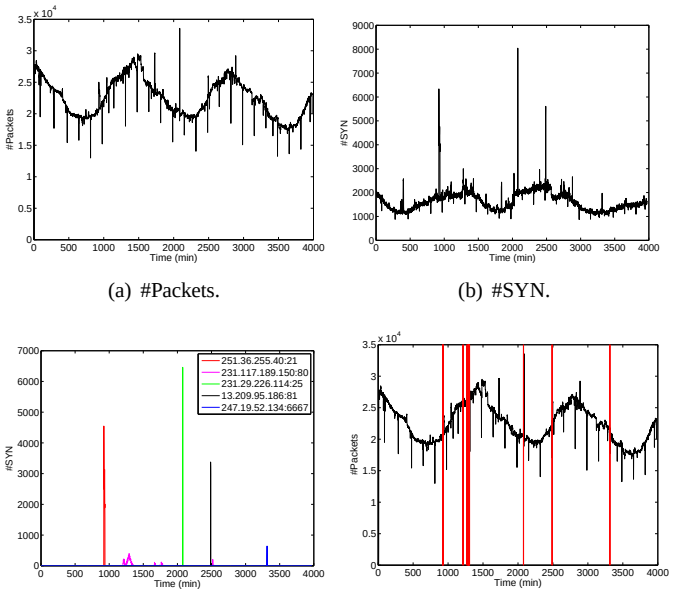
Non-parametric CUSUM is very efficient in detecting attacks even with low intensity, and with small average delay detection. In [10], it was proven that CUSUM is asymptotically optimal for minimizing average detection delay given a fixed false alarm rate.



(a) #SYN in Auckland traces. (b) #Packets in Auckland & mixed traces.

(c) #SYN in malicious traffic. (d) Raised alarms.

Fig. 4. Analysis results of mixed traces.



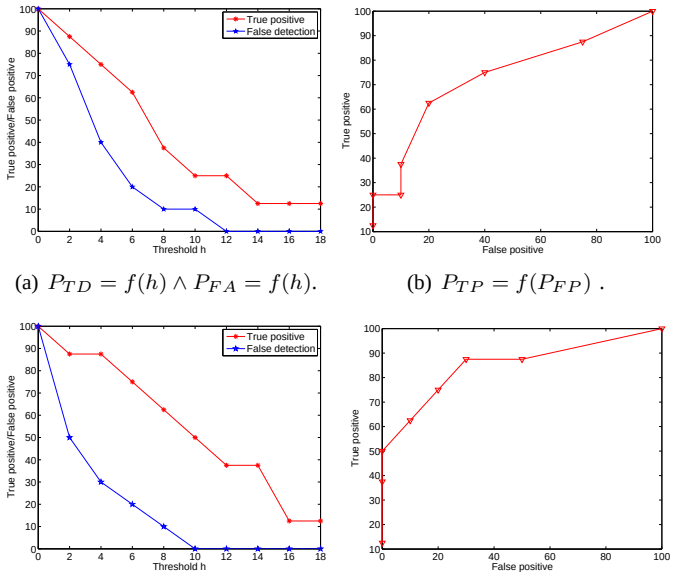
(a) #Packets. (b) #SYN.

(c) Detected malicious flows. (d) MNP-CUSUM raised alarms.

Fig. 5. Analysis results for OTIP trace.

B. Performance evaluation

We conduct performance analysis study via Receiver Operational characteristics (ROC) curve, to study the accuracy of the proposed framework. Our analysis verify the true positive probability vs. false positive probabilities ($P_{TP} = f(P_{FP})$), with the variation of the value of threshold h. These parameters are easily verified from mixed used traces, because we know in advance the start/stop instant of attacks, the IP address and port number of victim, and the number of existing attacks.



(a) $P_{TD} = f(h) \wedge P_{FA} = f(h)$. (b) $P_{TP} = f(P_{FP})$.

(c) $P_{TD} = f(h) \wedge P_{FA} = f(h)$. (d) $P_{TP} = f(P_{FP})$.

Fig. 6. P_{TD} and P_{FP} for $P = 12$ and $P = 14$.

P_{TD} is the number of detected attacks divided by the total number of existing ones. P_{FP} is the percentage of raised alarm that did not correspond to real attack. Fig. 6 illustrates the relation between true positive and false positive, as well as $P_{TD} = f(h)$ and $P_{FP} = f(h)$, where FP decreases as the threshold value increases, and true attacks may also completely missed. Hence, a tradeoff between false alarm and true positive detection is required to control sensitivity and prevent miss detection. We also notice that large sketch width decreases the false positive and increases the detection rate.

V. Conclusion

In this paper, we propose a new framework that integrates sketch and CUSUM for online anomalies detection at high speed link. Proposed framework is able to automatically pinpoint the malicious IP flows responsible of anomaly, through exploiting bucket index in an additional multi-layer sketch.

We proved the effectiveness of the proposed approach through implementation and testing at real traces with DDoS and scan attacks. Results of our experimentations have proved the capacity of early detection even for low intensity of DoS/DDoS attacks.

The proposed method is easily decentralized due to linear property of sketch with respect to addition operator. Future direction will be converged toward the hierarchical distribution of the proposed approach, and the reduction of the size of exchanged sketch information between different monitoring nodes in different layers.

Acknowledgments

This work has been partially funded by the French National Research Agency through the OSCAR project.

References

[1] M. Charikar, K. Chen, and M. Farach-Colton, "Finding frequent items in data streams," in *Proceedings of the 29th International Colloquium on Automata, Languages and Programming (ICALP '02)*. London, UK: Springer-Verlag, 2002, pp. 693–703.

[2] B. Krishnamurthy, S. Sen, Y. Zhang, and Y. Chen, "Sketch-based change detection: methods, evaluation, and applications," in *Proceedings of the 3rd ACM SIGCOMM conference on Internet measurement (IMC'03)*, New York, NY, USA, 2003, pp. 234–247.

[3] X. Li, F. Bian, M. Crovella, C. Diot, R. Govindan, G. Iannaccone, and A. Lakhina, "Detection and identification of network anomalies using sketch subspaces," in *Proceedings of the 6th ACM SIGCOMM on Internet measurement (IMC '06)*. New York, NY, USA: ACM Press, 2006, pp. 147–152.

[4] G. Cormode and S. Muthukrishnan, "An improved data stream summary: The count-min sketch and its applications," *Journal of Algorithms*, vol. 55, no. 1, pp. 58–75, April 2005.

[5] G. Cormode, F. Korn, S. Muthukrishnan, and D. Srivastava, "Diamond in the rough: Finding hierarchical heavy hitters in multi-dimensional data," in *Proceedings of the 23rd ACM SIGMOD*, 2004, pp. 155–166.

[6] Y. Li, J. Yang, C. An, and H. Zhang, "Finding hierarchical heavy hitters in network measurement system," in *Proceedings of the 2007 ACM symposium on Applied computing (SAC '07)*. New York, NY, USA: ACM Press, 2007, pp. 232–236.

[7] G. Cormode and S. Muthukrishnan, "What's new: Finding significant differences in network data streams," in *Proceedings of IEEE Infocom*, 2004, pp. 1534–1545.

[8] R. Schweller, Z. Li, Y. Chen, Y. Gao, A. Gupta, E. Parsons, Y. Zhang, P. Dinda, M.-Y. Kao, and G. Memik, "Reverse hashing for high-speed network monitoring: Algorithms, evaluation, and applications," in *Proceedings of IEEE International Conference on Computer Communications (INFOCOM 06)*, April 2006, pp. 1–12.

[9] H. Kim, B. Rozovskii, and A. Tartakovsky, "A nonparametric multichart cusum test for rapid intrusion detection," *International Journal of Computing and Information Science*, vol. 2, no. 3, pp. 149–158, December 2004.

[10] A. Tartakovsky, B. Rozovskii, R. Blazek, and H. Kim, "A novel approach to detection of intrusions in computer networks via adaptive sequential and batch-sequential change-point detection methods," *IEEE Transactions on Signal Processing*, vol. 54, no. 9, pp. 3372–3382, September 2006.

[11] R. Blazek, H. Kim, B. Rozovskii, and A. Tartakovsky, "The quickest sequential detection of intrusions in computer networks," in *Interface 2003*, Salt Lake City, Utah, March 2003.

[12] X. Li, F. Bian, M. Crovella, C. Diot, R. Govindan, G. Iannaccone, and A. Lakhina, "Detection and identification of network anomalies using sketch subspaces," in *Proceedings of the 6th ACM SIGCOMM on Internet measurement (IMC '06)*. New York, NY, USA: ACM Press, 2006, pp. 147–152.

[13] W. Feng, Z. Zhang, Z. Jia, and Z. Fu, "Reversible sketch based on the xor-based hashing," in *Proceedings of the Asia-Pacific Conference on Services Computing (APSCC '06)*, Guangzhou, Guangdong, China, December 2006, pp. 93–98.

[14] D. Wheeler and R. Needham, "TEA, a tiny encryption algorithm," in *Fast Software Encryption: Second International Workshop*, ser. LNCS, vol. 1008. Springer Verlag, 1999, pp. 363–366.

[15] S. Shepherd, "Tea source code," http://www.simonshepherd.supanet.com/source.htm.

[16] P. Gutmann, "Optimized rc4 code," http://www.zengl.net/freeswan/.

[17] Massive Data Analysis Lab: MassDal, "Count-min sketch source code," http://www.cs.rutgers.edu/7Emuthu/massdal-code-index.html.

[18] National Laboratory of Applied Network Research: NLANR, "Traces archive," http://pma.nlanr.net/Special/.

[19] V. A. Siris and F. Papagalou, "Application of anomaly detection algorithms for detecting syn flooding attacks," in *Proceedings of IEEE Global Telecommunications Conference (GLOBECOM '04)*, vol. 4, Dallas, USA, 2004, pp. 2050–2054.

[20] M. Thorup and Y. Zhang, "Tabulation based 4-universal hashing with applications to second moment estimation," in *Proceedings of the ACM-SIAM Symposium on Discrete Algorithms (SODA '04)*, New Orleans, Louisiana, USA, January 2004.