Making Flow-Based Security Detection Parallel

Marek Švepeš, Tomáš Čejka

Making flow-based security detection parallel

Marek Švepeš[1] and Tomáš Čejka[2]

[1] CTU in Prague, FIT
Thakurova 9, 160 00 Prague 6, Czech Republic
svepemar@fit.cvut.cz
[2] CESNET, a.l.e.
Zikova 4, 160 00 Prague 6, Czech Republic
cejkat@cesnet.cz

Abstract. Flow based monitoring is currently a standard approach suitable for large networks of ISP size. The main advantage of flow processing is a smaller amount of data due to aggregation. There are many reasons (such as huge volume of transferred data, attacks represented by many flow records) to develop scalable systems that can process flow data in parallel. This paper deals with splitting a stream of flow data in order to perform parallel anomaly detection on distributed computational nodes. Flow data distribution is focused not only on uniformity but mainly on successful detection. The results of an experimental analysis show that the proposed approach does not break important semantic relations between individual flow records and therefore it preserves detection results. All experiments were performed using real data traces from Czech National Education and Research Network.

1 Introduction

Flow-based monitoring plays a key role in network management. Not only it provides an overview of the traffic mix, it greatly helps with network security issues such as malicious traffic detection.

There are many types of malicious traffic that should be detected in real networks. As the speed and size of computer networks grow, it is necessary for network operators to process more and more data to be informed about the status of their network. However, with the increasing traffic volume, it is difficult to run lots of detection algorithms at once using just a single machine. The more data, the more computing resources are needed and the longer time the processing takes.

In order to overcome resource limits of a single machine, parallelism plays an important role. Various types of scalable architecture have been invented to process data in parallel. Generally, to be able to process more data, analyzer has to either run parts of its algorithms in parallel or split data for separate processing units.

Since the parallelization of individual detection algorithms is very dependent on the nature of the algorithm and, additionally, according to Amdahl's law,

there are parts of algorithms that can't be run in parallel, we have decided to focus on data distribution for independent processing units. Our aim is to split a continuous stream of network data (more specifically flow records, i.e. aggregated packet headers) into much smaller subsets that are being processed separately in parallel. We also focus on evaluation of the impact of data splitting on the security analysis results.

The contribution of this paper is to present our experiments with processing data traces from the real backbone network. The aim is to use existing algorithms from a single machine processing and deploy them in a distributed environment. This paper shows, that data splitting for such purpose is complicated due to semantic relations in data which should be preserved. Breaking the relations can cause that the obtained detection results are significantly worse than using a single processing machine. The paper also shows a feasible way how to split flow data with respect to semantic relations. Proposed approach preserves detection results and allows a scalable deployment.

This paper is organized as follows. Sec. 2 describes existing related work, i.e. systems for anomaly detection, traffic sampling and network traffic processing in parallel. Sec. 3 describes scattering methods that can be used to split flow records into a separate groups for parallel processing by independent computational nodes. Sec. 4 describes our testing environment that was created for our experiments. The section also presents results of measurement of described scattering methods. Sec. 5 concludes the paper.

2 Related work

This section describes related approaches of parallel network traffic analysis and anomaly detection usually done using Network Intrusion Detection System (NIDS) or Network Intrusion Prevention System (NIPS).

There are many existing systems for network traffic analysis and anomaly detection that are modular by design. For instance, TOPAS [1] and NEMEA [2] are flow-based systems that consists of modules that process data. When there is a big volume of flow data, running the systems on a single machine may reach resource limits of the machine. The systems do not support data distribution natively as it is available for various big data frameworks. However, NEMEA modules can be easily run and connected in a distributed environment.

K. Xinidis et al. in [3] presented an architecture with Active Splitter for distributed NIDS aiming for performance optimization of the detection sensors running Snort [4] (packet-based system performing deep packet inspection). The splitter uses hash functions for packet distribution and three techniques to optimize the performance of the sensors. Cumulative Acknowledgements reduce redundant sending of packets between splitter and sensors, Early Filtering in splitter applies Snort rule subset on packet headers (no payload inspection) and finally Locality Buffering reorders packets in a way that improves the locality of sensors memory accesses.

H. Sallay et al. in [5] made the network traffic analysis distributed using switch/router. The architecture contains dedicated sensors for individual services (e.g. FTP) and the incoming traffic is forwarded to them according to switching table of the switch/router. Sensors are running Snort but only with needed rule subset for their service. Since the volume of traffic of individual services can differ significantly, the load of computational nodes wouldn't be uniform. Therefore, this approach is not suitable for us.

Nam-Uk Kim et al. in [6] compare static and dynamic hash-based load balancing schemes and propose dynamic (i.e. adaptive) load-balancing scheme for NIDS. It uses a lookup table which is periodically reorganized according to historical packet distribution and current load of individual nodes. If needed, flows with the smallest volume are reorganized. Proposed method distributes packets in a way that does not break the flow stream, however, they don't take into account relations between individual flow records and the impact of splitting on detection results is not evaluated.

M. Valentin et al. in [7] presented a NIDS cluster for scalable intrusion detection. It consists of frontend nodes that distribute packets between backend nodes running Bro [8] for intrusion detection. Moreover, there are proxy nodes propagating state information of backend nodes and also one central manager node for collecting and aggregating results. Each frontend node distributes data from one monitored line and uses a hashing distribution scheme with a single hash function. The architecture requires backend nodes and proxy nodes to exchange data with detection subresults. In our approach, we are dealing with splitting a stream of flow records instead of packets. Our hashing distribution scheme is adjusted to provide all needed data to the detection methods for correct intrusion detection. Therefore, our computational nodes running intrusion detection are independent and don't communicate with each other. Finally, our proposed hashing distribution scheme represents a general way, how to split flow data with respect to detection results.

Big data frameworks such as Hadoop [9] or Spark [10] are distributed by nature. They are based on storage of data onto some distributed file system. A special designed parallel algorithm can be used to run on many distributed nodes and process all data. A universal and the most popular approach of distributed processing is MapReduce. However, the overall result of this kind of computation usually depends on the Reduce phase that merges local results from all nodes. Therefore, only low attention is paid to any relations or semantics during the data distribution and storage. An improved data distribution in Hadoop was presented as Hashdoop in [11]. Contrary, our approach is more general and it is applicable even on stream-wise processing with multiple different algorithms. Even though the main focus of ours is to make non-distributed system working in parallel, the principle described in our paper can be used for improvement of data distribution in big data frameworks as well.

Sampling has a common goal with parallel processing – capability to handle more data at the same time. J. Mai in [12] shows impact of the packet sampling on detection of portscanning and K. Bartos in [13] deals with flow sampling

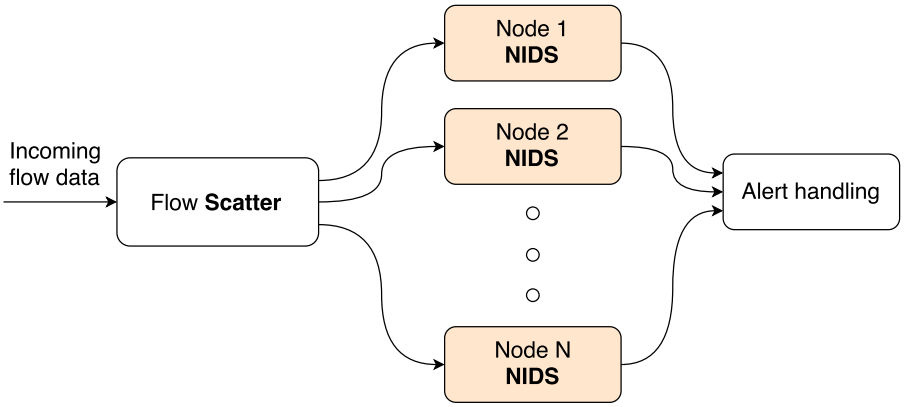

Fig. 1: A high level view of the infrastructure of scalable and distributed network flows analysis using NIDS.

techniques for anomaly detection. However none of these approaches can be applied on data splitting.

3 Flow distribution scheme

When designing a distribution scheme, several aspects have to be taken into account: i) the data should be distributed uniformly between all computational nodes, ii) the distribution algorithm should be as fast as possible in order to process as much data as possible, iii) the impact of splitting the data on detection results should be minimized.

In general, there are two ways how to distribute the data, statically or dynamically (also called static and dynamic load-balancing) and both have some pros and cons when applied in parallel NIDS. Static distribution has immutable rules for splitting the data e.g. a packet with source IP address 1.2.3.4 goes to node 1 and a packet with source IP address 5.6.7.8 goes to node 2. This preserves the data stream with possible security incident. However, it cannot affect the load of individual computational nodes when the distribution is not uniform. On the other hand dynamic distribution can perform some actions in order to make the load uniform (e.g. redirect some packets to less loaded computational nodes). Unfortunately, this behaviour can make the security incident invisible. Therefore, we have decided to use static distribution and focus on uniformity.

Fig. 1 shows high level view of the infrastructure of scalable and distributed network flows analysis using NIDS. The following subsections describe several splitting mechanisms used in the flow scatter.

3.1 Random scattering

Lets assume the detection results are not dependent on any semantic relations between flow data, i.e. scattering mechanism can distribute the data regardless

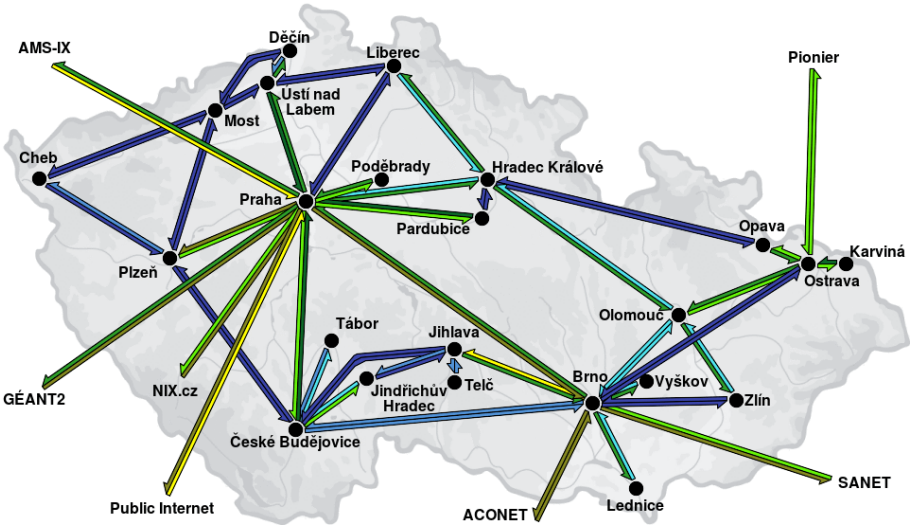

Fig. 2: Topology of Czech national research and education network (NREN) CESNET2, network traffic on the perimeter is analyzed.

of the information from flow records. In that case, the flow scatter can distribute the flow records between nodes using statistical uniform distribution, which is optimal for load-balancing. Received records by flow scatter are forwarded to computational nodes according to random number generator. It is clear that every random distribution splits the flow records into different subsets. However, as we discuss in Evaluation (Sec. 4), breaking semantic relations in flow data using random distribution affects the detection results.

3.2 Scattering based on network topology

Scattering based on network topology is another logical way of distributing the flow data. Since the computer networks are designed using hierarchical model that usually respects geographical and logical division into subnets and network lines.

Fig. 2 shows a high level topology of CESNET2 National Research and Education Network (NREN), which is a backbone academic network and it is also used as a transit network. It is inter-connected with other networks via several lines that are being monitored. The data taken from the monitoring probes contain a line identification — the line number. Flow scatter can easily distribute the data using these line numbers.

Standard monitoring infrastructure collects flow records from monitoring probes onto one central collector. In case of scattering based on network topology, this concept can be changed and it would be more efficient to send exported flow records directly to computational nodes.

3.3 Hash-based scattering

Hash functions are used to transform an input data into an output form with a fixed length. Cryptography expects that the output of an ideal hash function meets requirements such as uniform distribution and missing relation between output and input. In our case, the hash function can be used in the flow scatter to select an appropriate computational node number uniformly. Information from the incoming flow records can be used as an input for the hash function.

The dependency of selected node number on the input data of the hash function leads to divison of flow records into subsets with the same characteristics. The subsets with the same characteristics are then processed together on the same computational node and this can be used to preserve the detection results. For instance, if we use only the source IP address for hashing, all flow records having the same source IP address ends up on the same node. Meanwhile, flow records with different source IP addresses have a high probability to be processed with different nodes.

Let some set of flow records contain a security incident that can be detected using some detection method. Then, there exists a minimal subset of flow records with semantic relations that must be processed by this detection method together to get a correct result. In order to find the semantic relations in flow data, a set of detection methods was studied. The aim is to find a suitable set of information that is used as an input for hash function.

Studied detection methods

- Vertical SYN scanning can be detected using a threshold-based method published in [14]. To successfully detect this type of scanning, the method needs to receive all flow records of the same source IP address which is a possible attacker (scanner). Similar method can be used to detect horizontal SYN scanning. Source IP address is used for hashing.
- Brute-force password guessing against remote management services (SSH, TELNET, RDP etc.) can be detected using a method which needs to inspect all flow records between two IP addresses in both directions. An ordered pair of source and destination IP addresses (i.e. bi-flow) is used for hashing.
- There are many public lists of malicious addresses (black-lists). These addresses were abused due to various reasons like sharing malware, controlling botnets or acting in some anomalous evil way. Communication with a blacklisted IP address can indicate some malware infection and thus it should be reported. The detection is quite easy — every time any blacklisted address appears in a flow record, an alert can be sent. This type of detection is very efficient with a scattered data, because just a single flow record is needed to trigger an alert. Source IP address is used for hashing.
- More complex method based on statistics about IP addresses and matching the rules describing malicious traffic is able to detect DoS, DNS amplification, SSH brute-force password guessing and horizontal scanning. It needs to receive all flow records with the same IP address regardless of whether it is a

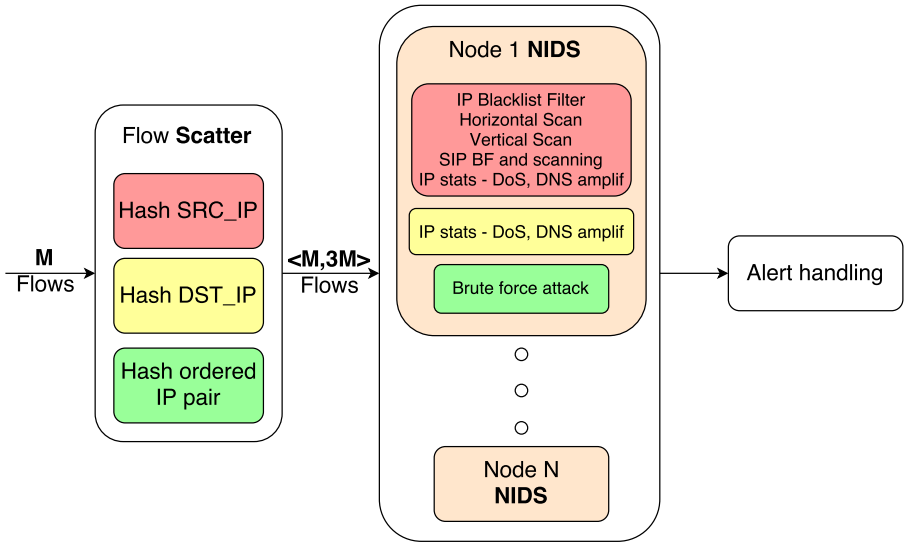

Fig. 3: Flow scatter contains three hash functions, each uses a specific information from flow records. The result of a hash function determines the computational node that processes the flow record with corresponding group of detection methods.

source or a destination IP. Therefore, hashing both source and destination IP addresses separately is needed in this case, which can result in duplication. The flow record can be forwarded to two different computational nodes. The duplication effect will be discussed later in this section.

– One of the detection methods based on application layer can detect brute-force attacks and scanning of user accounts on a Session Initiation Protocol (SIP) device. The detection method analyzes SIP responses from the server so all flows with the same source IP address must be delivered to the same node. Source IP address is used for hashing.

In general, we have recognized three groups of detection algorithms, whereas each group has to process all flow data with the same characteristic (e.g. same source IP address) on a single computation node. Therefore, we have a group of detection algorithms expecting all flow records with the same **source** address, a group expecting flow records with the same **destination** address and a group expecting flow records with the same **ordered pair of source and destination** addresses. Fig. 3 shows all three hash functions of the flow scatter where each hash function has the same color as the corresponding group of detection algorithms.

Since we want to run all detection algorithms in parallel, all three hash functions must be computed for every flow record. Naturally, results of the three hash functions can be different. Therefore, one flow record can be sent to at least one and, in the worst case, up to three computational nodes. This duplication

is caused by the number of different groups of algorithms and it is needed to provide all flow records that should be processed together to the algorithms (to preserve correct detection results).

In fact, the number of duplicates does not affect overall scaling of the parallel processing i.e. higher number of computational nodes does not increase the duplicates. Moreover, each hashing function determines a computational node, which processes the flow record with corresponding group of detection methods. Therefore, each group processes the flow record only on one computational node and every flow record is processed by all groups of detection methods. For example, if the selected nodes are 2 (for the SRC IP red hash) and 5 (for the DST IP yellow hash and for the IP pair green hash), it is processed by red group on `node 2`, yellow and green group on `node 5`. It means, that flow record may be duplicated, but only on a communication level between flow scatter and computational nodes.

To compare our approach with a single hashing function e.g. NIDS cluster [7] uses *hash = md5(srcIP + dstIP)*, we can show, that it would not work for us. Let's take methods for detection of horizontal port scanning and brute-force password guessing discussed in Sec. 3.3. The method for brute-force password guessing needs to see all flow records between source and destination IP addresses in both directions, so this hash function would work (*md5(A + B)* is equal *md5(B + A)*). On the other hand, horizontal scanning has the same source IP address but different destination addresses, so it is possible that two flow records with the same source IP but different destination IP could end up on a different computational node.

Our approach with multiple hash functions can be applied on arbitrary detection method. To do so, it is necessary to determine characteristics of needed flow data for correct detection result, as it was done in Sec. 3.3.

4 Experiments and Evaluation

In order to evaluate all important aspects of the scattering methods (uniformity, speed, impact on detection), the NEMEA system was chosen as a platform for our experiments and evaluation. The system itself has already implemented detection methods, which were studied and discussed in Sec. 3.3 and its efficient libraries allow us to process traffic from high speed backbone network. Overall, we have processed over 5 billions of flow records of real data traces in 10 different (pseudonymised) data sets captured in CESNET2 NREN during August and September 2016 with on average of 60,000 flows/s.

4.1 Testing environment

For our experiments we used a virtual machine with 64b Scientific Linux 7 OS, with the following hardware specification: 16 CPU cores, 24 GB RAM, 2 TB free disk capacity.

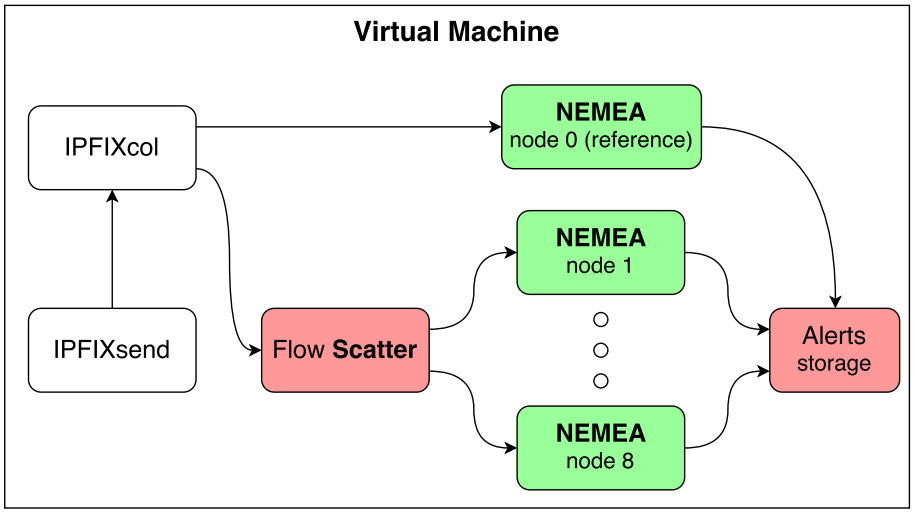

Fig. 4: Testing environment for experiments and evaluation of various methods of flow data distribution between computational nodes running flow-based NIDS NEMEA.

Fig. 4 shows the configuration of our testing environment. `IPFIXsend` and `IPFIXcol` [15] were used for replaying the IPFIX data in real-time. The flow data were received by the `flow scatter` and also directly by the `node 0` which was used as a reference single instance (it processes all flow data without any splitting). The `node 0` was a ground truth for us to evaluate an impact of data splitting on detection results. The flow scatter distributes flow data between nodes 1–8 as it was described earlier. All nodes contain exactly the same set of detection methods. During the experiments, we have collected data from 8 exporting probes that monitor different lines.

4.2 Results

The detection results from all nodes were stored and the analysis is described in this section.

Fig. 5 shows a comparison of an average distribution of flow records based on various scattering methods. The optimal value (red dashed line) is 12.5 % for 8 nodes. Random scattering achieves optimal results because of the used statistical uniform distribution. However, hash-based scattering is not significantly worse than the random (i.e. optimal) one. On the other hand, link-wise scattering is unbalanced because of different speeds of the monitored lines and the volume of traffic[3]. Node 1 even processed no data because there were no data exported

[3] We expect that such unbalanced distribution based on observation points can be observed in every network with lines of different bandwidth.

from the first line. For hash-based scattering, we have compared data distribution using two different hashing algorithms — CRC32 and Jenkins. On average, CRC32 had better results and therefore it was chosen as a final solution.

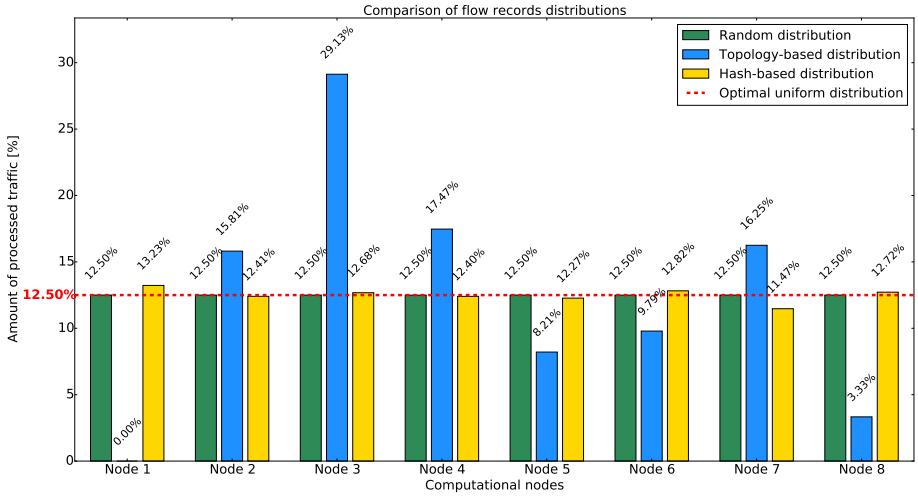

Fig. 5: Comparison of average flow records distributions using various scattering methods.

To analyze the reported alerts, we needed to compare the set of unique events from the reference node 0 with the set of unique events from all distributed nodes. To achieve this, the reported events of each detection method and each node were analyzed separately at first. Subsequently, the unique events were merged together. For example, in the case of horizontal scanning, if an attacker probes 50 or more computers in two different subnets, where 50 is a threshold of the detection algorithm, 2 events should be reported. Hash-based scattering delivers all flow records representing this traffic to the same node due to the source IP address hashing. Using link-wise scattering, the flow records could end up on different nodes because the traffic can go through different lines. Random scattering will split the flow records randomly.

Fig. 6 shows a comparison of the detection results after applying various scattering methods. Note, that *Hoststatsnemea* in the figure legend stands for the method based on statistics about IP addresses, which was discussed in section 3.3. The first column represents the reference instance with 100 % reported events, whereas each type of events has a different color and it is normalized so that the number of different event types are represented equally. Random distribution (the second column) has a huge impact on the detection results because of breaking the semantic relations between flow records. This was an expected result, however, the random distribution is a reference of optimal flow data distribution. Scattering based on the network topology (the third column) caused

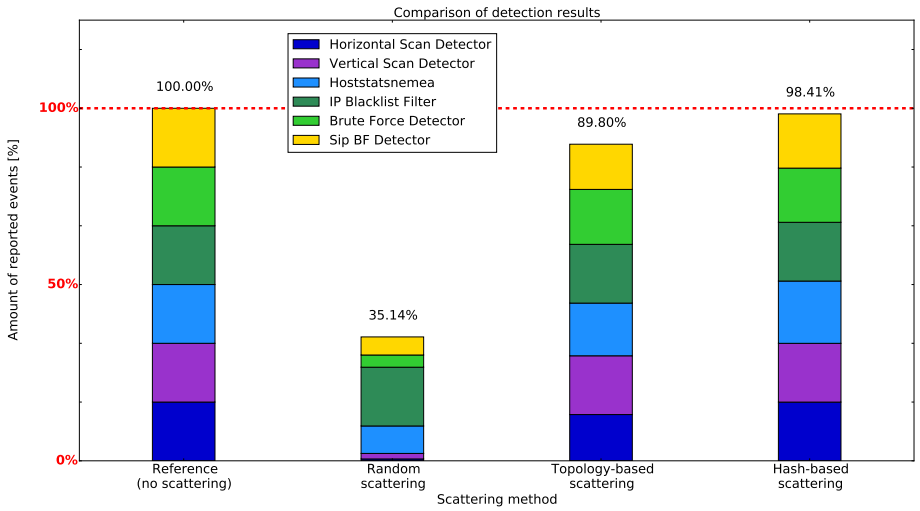

Fig. 6: Comparison of the detection results after applying various scattering methods. Each part of column with different color stands for normalized number of unique events reported by different detection method.

that some of distributed attacks and, in general, N:1 or 1:N attacks (DDoS, horizontal scanning etc.) were not detected. The last column shows that scattering based on hashing specific information from flow data has the best results. The reason of undetected events is probably a periodic clean-up of structures containing information and timing of stream-wise detection algorithms.

After the evaluation of the uniformity and the impact on the detection, we tested a maximal throughput of the hash-based flow scatter as the best method for distribution. A simple NEMEA module was created to generate and send 100 million flow records at full speed to the flow scatter. Measured computation time was focused on the main cycle receiving the flow record, hashing, making decision about number of computational nodes the flow belongs to according to the computed hashes and sending the flow record. The maximal throughput was on average 1.8 million flow records per second.

5 Conclusion

This paper presented the results of practical experiments with different approaches of splitting a stream of network flow data for the purposes of parallel anomaly detection. The aim of our work was to compare not only a uniformity of distribution but also an impact of data splitting on the detection results. Our experiments were performed using real traffic traces from Czech national research and education network (NREN). For simulation of parallel processing, we used an open source detection system NEMEA, however, the analysis results

are general enough and we believe that the proposed distribution approach can be used with any other detection system.

We have recognized three groups of detection algorithms with different requirements on data. Therefore, we have designed a flow scatter that uses three different hashing specific information from flow records (source address, destination address, ordered pair of source and destination address) to provide all needed data to independent computational nodes. The results of our experiment show that our approach preserves semantic relations in flow data that are important for different groups of detection algorithms and therefore the results of parallel detection are similar to reference results without splitting the data.

With the proposed approach of flow data distribution, it is possible to use detection methods that are deployed on a single machine and run them in parallel without changes. As a future work, we want to make more experiments with scaling beyond the measured throughput of the flow scatter by using multiple flow scatters in parallel and distribute incoming flow records between the flow scatters with e.g. round robin.

Acknowledgment

This work was supported by the Technology Agency of the Czech Republic under No. TA04010062 *Technology for processing and analysis of network data in big data concept*, grant No. SGS17/212/OHK3/3T/18 funded by MEYS and the project Reg. No. CZ.02.1.01/0.0/0.0/16_013/0001797 co-funded by the MEYS and ERDF.

References

1. Munz, G., Carle, G.: Real-time analysis of flow data for network attack detection. In: 2007 10th IFIP/IEEE International Symposium on Integrated Network Management. (May 2007) 100–108
2. Cejka, T., Bartos, V., Svepes, M., Rosa, Z., Kubatova, H.: Nemea: A framework for network traffic analysis. In: 2016 12th International Conference on Network and Service Management (CNSM). (October 2016) 195–201
3. Xinidis, K., Charitakis, I., Antonatos, S., Anagnostakis, K.G., Markatos, E.P.: An active splitter architecture for intrusion detection and prevention. IEEE Transactions on Dependable and Secure Computing **3**(1) (January 2006) 31–44
4. Roesch, M.: Snort - lightweight intrusion detection for networks. In: Proceedings of the 13th USENIX Conference on System Administration. LISA '99, Berkeley, CA, USA, USENIX Association (1999) 229–238
5. Sallay, H., Alshalfan, K.A., Fred, O.B., Words, K.: A scalable distributed ids architecture for high speed networks. In: IJCSNS International Journal of Computer Science and Network Security, VOL.9 No.8, Citeseer (2009)
6. Kim, N., Jung, S., Chung, T.: An efficient hash-based load balancing scheme to support parallel NIDS. In: Computational Science and Its Applications - ICCSA 2011 - International Conference, Santander, Spain, June 20-23, 2011. Proceedings, Part I, Springer (January 2011) 537–549

7. Vallentin, M., Sommer, R., Lee, J., Leres, C., Paxson, V., Tierney, B.: The NIDS cluster: Scalable, stateful network intrusion detection on commodity hardware. In: Recent Advances in Intrusion Detection: 10th International Symposium, RAID 2007, Gold Goast, Australia, September 5-7, 2007. Proceedings, Berlin, Heidelberg, Springer Berlin Heidelberg (2007) 107–126
8. Paxson, V.: Bro: A system for detecting network intruders in real-time. Comput. Netw. **31**(23-24) (December 1999) 2435–2463
9. Apache: Hadoop http://hadoop.apache.org.
10. Apache: Spark http://spark.apache.org.
11. Fontugne, R., Mazel, J., Fukuda, K.: Hashdoop: A MapReduce framework for network anomaly detection. In: IEEE Conference on Computer Communications Workshops (INFOCOM). (2014)
12. Mai, J., Sridharan, A., Chuah, C.N., Zang, H., Ye, T.: Impact of packet sampling on portscan detection. IEEE Journal on Selected Areas in Communications **24**(12) (December 2006) 2285–2298
13. Bartos, K., Rehak, M.: Towards efficient flow sampling technique for anomaly detection. In: Proceedings of the 4th International Conference on Traffic Monitoring and Analysis. TMA'12, Berlin, Heidelberg, Springer-Verlag (2012) 93–106
14. Cejka, T., Svepes, M.: Analysis of vertical scans discovered by naive detection. In: Management and Security in the Age of Hyperconnectivity: 10th IFIP WG 6.6 International Conference on Autonomous Infrastructure, Management, and Security, AIMS 2016, Munich, Germany, Springer International Publishing (2016) 165–169
15. Velan, P., Krejčí, R.: Flow information storage assessment using ipfixcol. In: Proceedings of the 6th IFIP WG 6.6 International Autonomous Infrastructure, Management, and Security Conference on Dependable Networks and Services. AIMS'12, Berlin, Heidelberg, Springer-Verlag (2012) 155–158