Flexible Flow Aggregation for Adaptive Network Monitoring

Falko Dressler
Autonomic Networking Group
Dept. of Computer Science 7
University of Erlangen, Germany
dressler@informatik.uni-erlangen.de

Gerhard Münz
Computer Networks and Internet
Wilhelm Schickard Institute for Computer Science
University of Tübingen, Germany
muenz@informatik.uni-tuebingen.de

Abstract

Network monitoring is a major building block for many domains in communication networks. Besides typical accounting mechanisms and the emerging area of charging in next generation networks, especially network security solutions rely on efficient and optimized monitoring. Network monitoring in high-speed networks is usually based on flow accounting and aggregation techniques represent a necessary enhancement in order to cope with increasing amounts of monitoring data that accrue with the ever-growing network capacities. In this paper, we propose a flexible flow aggregation mechanism that can be directly employed on a monitoring probe to reduce the memory and processing demands. Alternatively, it can work as a concentrator that collects flow data from multiple monitoring probes, combines and aggregates them and forwards the results to an analyzer. We verified and evaluated the aggregation mechanism by integrating it into our monitoring probe Vermont. Our approach opens new prospects for high-speed network monitoring and allows coping with special situations that cannot be treated satisfyingly by traditional flow accounting, such as distributed denial-of-service attacks causing very high numbers of flows. Aggregated flow data are an easy-to-handle form of packet information especially for anomaly detection and accounting issues.

1. Introduction

Network monitoring has become an important research area since the development of the first computer networks. At the beginning, monitoring data were used for network maintenance. In this context, network operators employed network monitoring techniques [1] in order to verify the deployed routing strategies or to locate potential bottlenecks for sufficient re-dimensioning of the networks [6]. Soon, additional application domains appeared such as accounting and the detection of suspicious activities such as attacks, propagating worms etc [10]. While the aforementioned applications are still relevant, other domains also require appropriate network monitoring such as intrusion detection and traceback mechanisms [13] or charging applications for next generation networks [7]. Manufacturers have responded by developing specific monitoring devices or by adding monitoring functionality into existing products like routers or firewalls.

A common monitoring technique is flow accounting. The key idea of flow accounting is to store information about packet flows and the corresponding statistics instead of individual packet information. In this context, a flow is defined as a unidirectional stream of IP packets identified by a common IP-five-tuple (protocol type, source IP address, destination IP address, source port, destination port). To this end, a single measurement record can contain information on up to several thousand packets. For transmission of monitoring data to a remote analyzer, a series of protocols named Netflow were developed by Cisco. The latest version Netflow.v9 has been documented as informational RFC [2]. Its successor is the IPFIX protocol [3] and the corresponding information model [15], both developed by the IPFIX working group at the IETF. First open-source implementations that support Netflow.v9 as well as IPFIX are available, e.g. nprobe [4], NetMate [19], and Vermont [11].

Although flow accounting works well under normal conditions (typical connections consist of about 7.7 packets per flow [12]), there is a major problem during DDoS (Distributed Denial of Service) attacks [14]. Many attacks randomly forge the source addresses of the attack packets, which results in an enormous number of different IP-five-tuples, each constituting an individual flow. As a consequence, many flows get lost because of limited resources at the monitoring probe, and the processed flows cause a huge amount of mostly useless data that is sent to the analyzer. Moreover, the analyzer will face a serious performance deficiency in processing all the received monitoring data.

To cope with this problem, flow sampling or flow aggregation schemes need to be employed. Flow sampling uti-

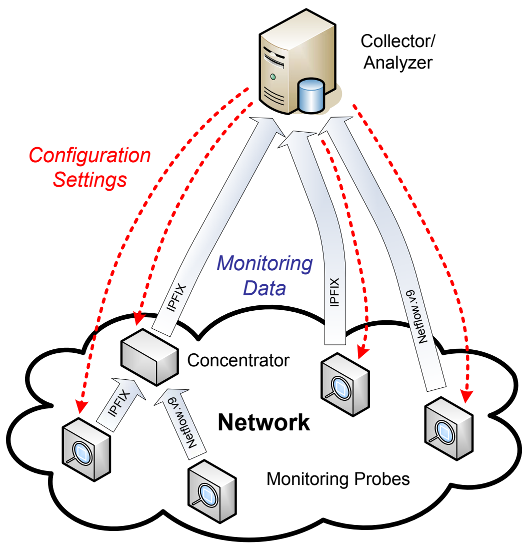


Figure 1. Adaptive network monitoring

lizes similar sampling and filtering algorithms as proposed in PSAMP but applies them to flows instead of packets [8]. Adaptive flow aggregation has been addressed by Hu et al. [9], proposing to adapt the aggregation level dynamically according to the available resources of the monitoring probe. Although this approach avoids flow loss in DDoS attack situations as described above, it does not take into account that arbitrarily defined flow aggregates usually do not meet the requirements of the analyzer. If, on the other hand, the analyzer was satisfied with aggregated flow data, the corresponding aggregation could be constantly applied.

In this paper, we introduce an aggregation mechanism that performs flow aggregation in a configurable manner. This aggregation mechanism allows adapting the amount of exported monitoring data to the current needs and available resources of the analyzer. Flexibility is obtained by rule-based control of the aggregation process, thus providing the basic functionality to build a control-loop between monitoring probe and analyzer as sketched in figure 1. With example configurations, we demonstrate the applicability and the advantages of our approach. Furthermore, we evaluated the capability to reduce the amount of flow data by monitoring two typical network links: the network connection of our group server and the Internet connection of the University of Erlangen.

The remainder of the paper is organized as follows. The proposed aggregation mechanism is discussed in section 2. The implementation and evaluation of the method is presented in section 3. A discussion in section 4 concludes the paper.

2. Flow Aggregation

2.1. Reasons for Introducing Flow Aggregation Techniques

Usually, a flow is defined as a set of packets that share the same IP-five-tuple information and that are observed within a given time interval at a specific observation point in the network. Distinguishing flows on the basis of the IP-five-tuple information results in a large amount of monitoring data that has to be exported by the monitoring probe, and later received and processed by the analyzer. In order to limit the amount of exported monitoring data, commercial monitoring probes often export only flow records with high traffic volume. Another approach is to apply sampling algorithms on the generated flow records that – similar to packet sampling techniques – select a subset of all generated flow records for export [8]. On the other hand, the fine IP-five-tuple granularity is not required by many applications processing the monitoring data. Such a typical application is accounting, where the total traffic volume sent and received by a customer's host has to be determined for billing purposes. Reducing the amount of monitoring data is also required if the data is to be stored. Note that already in midsize enterprise networks, one or more Gigabytes of flow records are easily generated in a single day[1].

Flow aggregation techniques reduce the amount of monitoring data by discarding unneeded flow information and merging multiple flow records with similar properties. As a result, the aggregated flow information describes the same traffic observation at a coarser level of granularity and with fewer details. The following list gives some examples:

- A common way to reduce the amount of monitoring data is to reduce the time resolution by merging subsequent records of the same flow into one record covering a larger time interval. This technique is often deployed if the monitoring data has to be archived.

- In case of client-server applications, information about the requested service is mostly contained in the port number at the server side. The client port number is usually dynamically assigned by the operating system of the client host. Hence, client port numbers can be discarded from the flow information without loss of valuable information.

- As mentioned above, accounting applications usually require information about the traffic volume sent and received by a connected client or exchanged at a gateway between two peering networks. Information about every single flow is not required. Especially, source

[1]In the backbone network of the University of Tübingen, about 10GB per day of Netflow data are generated.

and/or destination addresses and port numbers are not considered and can be discarded.

Currently, flow aggregation techniques are largely deployed at the analyzer. Depending on the needs of the application, the monitoring data is compressed and unneeded information is discarded. However, flow aggregation performed at the analyzer does not reduce the amount of monitoring data generated by the monitoring probe and transmitted to the analyzer. To circumvent this problem, we propose a configurable aggregation scheme that is deployed at the monitoring probe before the monitoring data is exported.

2.2. IPFIX Aggregation at the Monitoring Probe

In an RFC summarizing the IPFIX requirements [16], the IPFIX working group at the IETF has identified the necessity of implementing flow aggregation functionality in the monitoring probes. This idea is picked up in the IPFIX architecture draft [17] where an example shows how flows can be aggregated after masking the source and/or destination IP addresses. In [5], we presented how the required functionality can be realized and implemented. We further proposed an extension to the IPFIX protocol that allows exporting aggregated flow information in an efficient way. In the following subsections, our approach will be explained in more detail.

2.3. Aggregation Process

In accordance with [16], the proposed aggregation functionality can be realized either as an additional process that is installed between the metering process(es) and the exporting process of a monitoring probe, or within a separate device that we call a concentrator (see figure 1). Such a concentrator receives flow records from other monitoring probes, performs the flow aggregation, and exports the aggregated flow information to an analyzer. Both options are depicted in figure 2.

In order to differentiate between regular flow records and records containing aggregated information, we introduce the term flow aggregate. In general, a flow aggregate comprises a set of individual IP-five-tuple flows that share some common properties, the so called flow keys. The corresponding flow record contains information that identifies and describes the flow aggregate as a whole.

Which and how individual flows are aggregated into flow aggregates is specified by aggregation rules. An aggregation rule specifies which fields (flow keys and non-flow keys) are kept and which fields are discarded in order not to appear in the resulting flow record any more. For IP addresses, a mask operation can be applied in order limit the distinction between different flow aggregates to network addresses. Sometimes, the application of an aggregation rule is restricted to flows that match a given criteria. For example, to aggregate flows originating from a specific host or network, a selection pattern for the corresponding field must be specified.

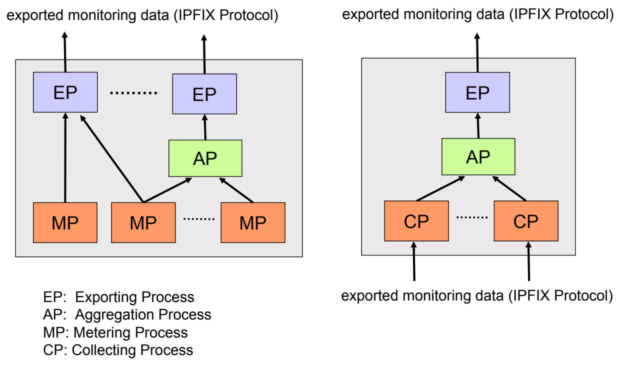

exported monitoring data (IPFIX Protocol)
exported monitoring data (IPFIX Protocol)
EP
EP
AP
MP
MP
MP
EP
AP
CP
CP
exported monitoring data (IPFIX Protocol)
EP: Exporting Process
AP: Aggregation Process
MP: Metering Process
CP: Collecting Process

Figure 2. Architecture of IPFIX aggregation. Aggregation can be employed within a monitoring probe (left) or in form of a separate concentrator (right)

Fields carrying measurement information are aggregated in a special way in order to get the corresponding values for the flow aggregate. As an example, packet and byte counters of the individual flows are summed up, while start and end time of the flow measurement are set to the minimum and maximum of all aggregated flows respectively.

An aggregation rule defines the treatment of the considered fields in separate lines according to the following structure:

1. A field modifier (discard, keep, mask/n, or aggregate) specifies how this field is treated and implicitly defines if the field appears in the resulting flow record or not;

2. the considered field identified by the corresponding information element ID [15];

3. an optional pattern that restricts the aggregation to flows or packets that match this pattern.

An aggregation rule is only applicable to an incoming flow or packet if all specified fields are present and if all indicated patterns match.

Multiple aggregation rules can be specified to generate flow aggregates with different properties. Incoming flow records are usually checked against each rule and may thus contribute to various flow aggregates at a time. In some cases, however, it is preferred that an incoming flow record that contributes to one flow aggregate is excluded from other flow aggregates. This can be achieved by organizing aggregation rules in chains or trees that define in which order the rules are to be checked. According to the rule semantic defined in [5], only the first matching rule in a chain or tree is applied.

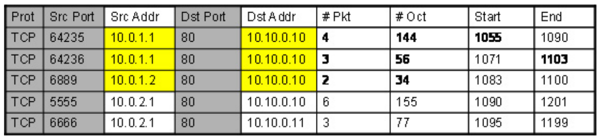
2.4. Aggregation Example

We demonstrate the principles and advantages of our flow aggregation mechanisms with the following aggregation rule example:

```
1  Aggregation rule
2    discard protocolIdentifier in TCP
3    discard sourceTransportPort
4    mask/24 sourceIpv4Address
5    discard destinationTransportPort in 80,443
6    keep destinationIpv4Address in 10.10.0.0/16
7    aggregate packetDeltaCount
8    aggregate octetDeltaCount
9    aggregate flowStartMilliSeconds
10   aggregate flowEndMilliSeconds
```

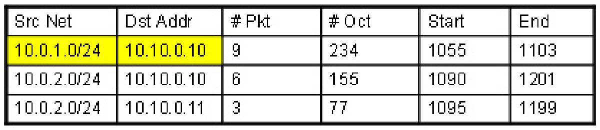
The first line indicates the beginning of the aggregation rule. Then (lines 2 and 3), two fields (transport protocol and source port) are specified that have to be present in each flow record to be aggregated. However, the resulting flow record will not include these fields because of the modifier `discard`. The pattern `in TCP` restricts the rule to TCP flows. The next line (4) requests the source address to be present in received records. This address field will be masked to 24 bit before aggregating (`mask/24`). The `discard` modifier is used again in line (5) but an additional pattern is specified which must be matched by received flows. Similarly, in (6) a pattern is provided but the `keep` modifier defines that this field appears in the flow record. Finally (6-10), the aggregate modifier is used to collect statistics of the flow such as the number of packets/octets and the flow start and end time.

Figure 3 shows some non-aggregated flow data. If the presented aggregation rule is applied, the flow aggregates shown in figure 4 will be created. Both, the number of flows as well as the number of fields in each flow is reduced. This reduction obviously depends on the kind of monitoring data (address/port distribution etc.) and the configuration of the aggregation rules. The proposed aggregation scheme provides a flexible mechanism to adapt the granularity and quantity of the aggregation depending on the particular scenario. Some sample measures are provided in section 3.



Prot	Src Port	Src Addr	Dst Port	Dst Addr	# Pkt	# Oct	Start	End
TCP	64235	10.0.1.1	80	10.10.0.10	**4**	**144**	**1055**	1090
TCP	64236	10.0.1.1	80	10.10.0.10	**3**	**56**	1071	**1103**
TCP	6889	10.0.1.2	80	10.10.0.10	**2**	**34**	1083	1100
TCP	5555	10.0.2.1	80	10.10.0.10	6	155	1090	1201
TCP	6666	10.0.2.1	80	10.10.0.11	3	77	1095	1199

Figure 3. Non-aggregated flow data consisting of protocol information, source and destination address, source and destination port, and statistical data



Src Net	Dst Addr	# Pkt	# Oct	Start	End
10.0.1.0/24	10.10.0.10	9	234	1055	1103
10.0.2.0/24	10.10.0.10	6	155	1090	1201
10.0.2.0/24	10.10.0.11	3	77	1095	1199

Figure 4. Resulting flow aggregates corresponding to the given rule set

2.5. Export of Flow Aggregate Information

Since IPFIX is a unidirectional protocol that does not allow querying specific information from the monitoring probe, the exported monitoring data should be self-explanatory, i.e. the analyzer should be able to understand and interpret the received information without any additional knowledge. In case of flow aggregates, the analyzer has to be informed about the corresponding aggregation rules. Otherwise, the circumstances under which the flow records were generated would not be clear. Especially, if aggregation rules contain patterns that restrict their application to a selection of incoming flow records, the analyzer has to be informed that the received flow information is biased. Moreover, if rules are organized in chains or tree as explained in the previous subsection, the order in which rules are checked also has to be disclosed.

We solved this problem by specifying a new template type called *data template*. A data template describes the structure of flow records and includes additional information about the patterns used in the aggregation rules. In practice, a separate data template is used for each aggregation rule. A special field in the data template serves as a backward pointer to a preceding rule if the aggregation rules are organized in a chain or tree. Detailed information about the data template can be found in [5].

3. Implementation and Evaluation

3.1. Implementation in Vermont

We used our modular IPFIX/PSAMP probe Vermont[2] to implement the presented aggregation mechanism. The resulting architecture of Vermont is shown in figure 5. Using this architecture, it is possible to use the aggregation mecha-

[2]Versatile Monitoring Toolkit, http://vermont.sourceforge.net/

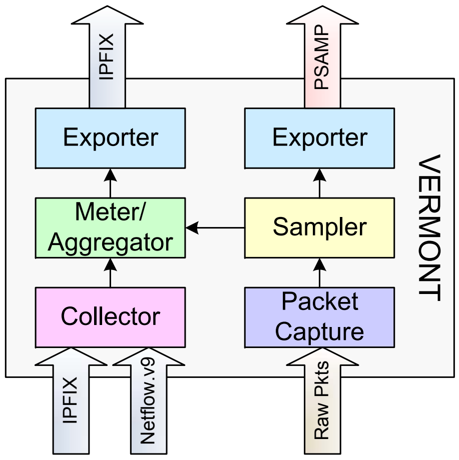


Figure 5. Architecture of Vermont including the aggregation module

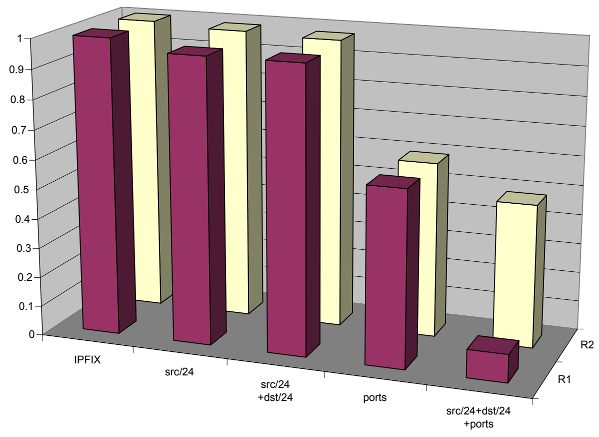


Figure 6. Aggregation ratio in comparison to flow accounting without aggregation. R1: workgroup server, R2: Internet connection

nism in both ways as mentioned before: directly on a monitoring probe before exporting the flow data or in form of a concentrator that receives flow records (either Netflow.v9 or IPFIX) and exports flow aggregate information. The first path is to capture packets using the pcap library, optionally filtering the collected data, and then sending the packets to the aggregation module using the 'hook' as shown in the figure. Alternatively, a collector module can decode Netflow and IPFIX data and put the received flow information directly into the aggregation module. The export of IPFIX conform flow data is provided by a separate exporter library. We verified the interoperability of Vermont with other IPFIX/Netflow exporters and collectors at an IPFIX interoperability event in 2005 [18].

3.2. Experimental Analysis

Even though the compression degree of the aggregation mechanism strongly depends on the properties of the monitored traffic as well as on the aggregation rules in use, we did some measurements in order to estimate the capabilities of our aggregation mechanism. The experiments were done using a standard PC (2.8GHz) running FreeBSD. Vermont was used for monitoring network data received at a dedicated mirror port of a Gigabit switch. The first experiment was done in front of our workgroup server where usually only small traffic volumes are observed (e-mail transfers and remote logins). The second experiment was done at the Internet gateway of the University of Erlangen (actually 622MBit/s). Both experiments lasted for 30 minutes in order to keep the amount of monitoring data at a manageable size. We evaluated the aggregation efficiency using five different aggregation schemes:

- Standard IPFIX export (IP-five-tuple flows) without aggregation (labeled IPFIX)
- Aggregation of flows from the same source network (src/24)
- Aggregation of flows from the same source network destined to the same destination network (src/24+dst/24)
- Aggregation of flows without considering port information (ports)
- Combination of the last two aggregation schemes (src/24+dst/24+ports)

The aggregation ratio of all mentioned aggregation schemes and for both experimental setups is shown in figure 6. Obviously, the aggregation of network addresses seems to be inefficient with respect to data compression (the aggregation ratio is about 0.97). This effect occurs due to different values in the source and destination ports. If these values are ignored, the aggregation ratio increases (ports,about 0.6), especially if combined with aggregation of address information (src/24+dst/24+ports). In the last case, we achieve an aggregation ratio of 0.48 on our Internet connection and down to 0.09 in the workgroup server scenario.

The two plots shown in figures 7 and 8 depict the packet rate of the monitoring data sent to the analyzer. Even if this measure hides possible overhead due to packet encapsulation, it is a valid basis if a continuous process is observed

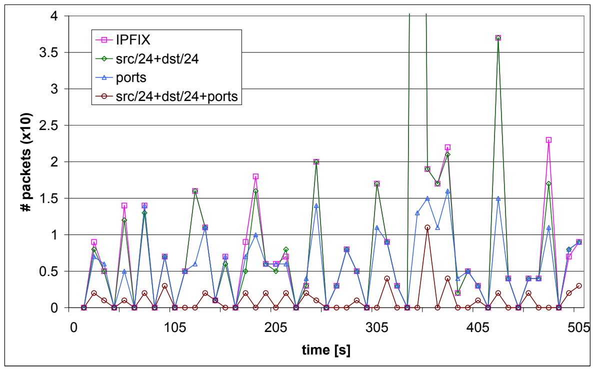


Figure 7. Rate of flow data as sent from the monitor to an analyzer: workgroup server (the average observed input rate is 32 packets/s)

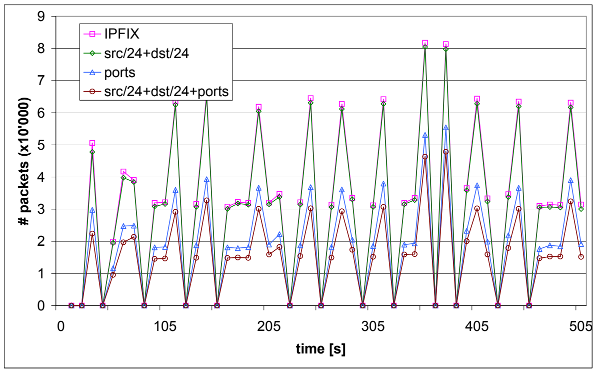


Figure 8. Rate of flow data as sent from the monitor to an analyzer: Internet connection (the average observed input rate is 58'500 packets/s)

or high data rates must be collected. Both figures demonstrate the reduced amount of monitoring data sent from the monitoring probe to the analyzer. Not shown is the reduction of the load of the monitor itself. Because many flows are mapped to a single flow aggregate, the corresponding hash table sizes are reduced. As a consequence, the memory requirements of the monitoring probe can be decreased and the search for existing flow records requires fewer steps, such fewer CPU cycles.

3.3. Application Scenarios

In this subsection, we show how the aggregation mechanism can be practically deployed in three different scenarios: accounting, anomaly detection, and TCP SYN flood detection.

Typical accounting solutions intend to associate consumed bandwidth or time to individual end systems. Exact descriptions about all monitored flows are not required. Therefore, the following rule set is appropriate to aggregate transmitted octets from/to each host in the monitored network (10.10.0.0/16) resulting in two flow aggregates for each host (incoming/outgoing):

```
Aggregation rule
  keep sourceIpv4Address in 10.10.0.0/16
  aggregate octetDeltaCount
  aggregate flowStartMilliSeconds
  aggregate flowEndMilliSeconds
Aggregation rule
  keep destinationIpv4Address in 10.10.0.0/16
  aggregate octetDeltaCount
  aggregate flowStartMilliSeconds
  aggregate flowEndMilliSeconds
```

Using this rule set, the aggregation process provides an efficient preprocessing of the monitored data. Therefore, flow aggregation also reduces the computational resources at the analyzer, i.e. the accounting process in this scenario.

Attack and anomaly detection have different objectives. Usually, the general behavior of all observed flows is considered and not individual flows. For example, the total number of observed packets might be of interest (rule 1), the number of monitored ICMP packets (rule 2), the distribution of traffic to different subnets (rule 3, 4), and the number of packets sent to request a specific service, e.g. web services (rule 5).

```
Aggregation rule 1
  aggregate packetDeltaCount
  aggregate flowStartMilliSeconds
  aggregate flowEndMilliSeconds
Aggregation rule 2
  discard protocolIdentifier in ICMP
  aggregate packetDeltaCount
  aggregate flowStartMilliSeconds
  aggregate flowEndMilliSeconds
Aggregation rule 3
  mask/24 sourceIpv4Address in 10.10.0.0/16
  aggregate packetDeltaCount
  aggregate flowStartMilliSeconds
  aggregate flowEndMilliSeconds
Aggregation rule 4
  mask/24 destinationIpv4Address in 10.10.0.0/16
  aggregate packetDeltaCount
  aggregate flowStartMilliSeconds
  aggregate flowEndMilliSeconds
Aggregation rule 5
  discard destinationTransportPort in 80,443
  aggregate packetDeltaCount
  aggregate flowStartMilliSeconds
  aggregate flowEndMilliSeconds
```

Similarly to the accounting scenario, preprocessing is provided by the aggregation process. The resulting flow data can be directly feed into the corresponding anomaly detection engines of the attack or intrusion detection system.

In the European project Diadem Firewall[3], we employ the following rule set on a monitoring probe for counting TCP SYN packets per destination and TCP SYN/ACK packets per source:

```
Aggregation rule
  discard protocolIdentifier in TCP
  keep destinationTransportPort
  keep destinationIpv4Address
  discard tcpControlBits in SYN
  aggregate packetDeltaCount
  aggregate flowStartSeconds
  aggregate flowEndSeconds
Aggregation rule
  discard protocolidentifier in TCP
  keep sourceIpv4Address
  keep sourceTransportPort
  discard tcpControlBits in SYN,ACK
  aggregate packetDeltaCount
  aggregate flowStartSeconds
  aggregate flowEndSeconds
```

Note that although protocol and TCP control bit fields are discarded from the flow records, the values of the patterns (TCP, SYN, SYN/ACK) are exported with the data templates. The resulting packet counters serve as input to a detection algorithm for TCP SYN flood attacks [20]. Under normal conditions, every SYN packet sent to a certain destination should be answered by a SYN/ACK packet, apart from some SYN packets that are directed to closed ports or non-existing hosts. During a TCP SYN flood attack, many SYN packets are directed to a single open port at the victim host. Returned SYN/ACK packets remain unanswered resulting in many half-open TCP connections. A victim host without specific protection mechanism is able to handle a limited number of half-open TCP connections only. If this limit is reached, new SYN packets are not answered by SYN/ACK packets any more, which is reflected by a high difference in the exported counter values. The number of flow records resulting from the above rule set depends on the number of attacked *(address, port)* pairs but not on the number of attack sources. This is important because address spoofing is usually applied in a SYN flood, causing a huge number of different IP-five-tuples.

4. Discussion and Conclusions

In this paper, we proposed an aggregation mechanism for efficient and flexible use in flow accounting scenarios. This mechanism is controlled by aggregation rules that allow adapting to particular application requirements. We specified the necessary functionalities as enhancements of the IPFIX architecture and protocol. The aggregation mechanism can be deployed in two ways: either at the monitoring probe to reduce the number of flow directly at the observation point, or in a separated device called concentrator. The second option allows merging and aggregating flow information produced at different observation points in the network into a single stream that is sent to the analyzer. The aggregation mechanism supports multiple active aggregation rules at a time as well as patterns that provide a filtering functionality.

We exemplarily evaluated the aggregation mechanism by using an implementation of the aggregation module for the monitoring toolkit Vermont. The experiments shows the data compression capabilities of different aggregation rules. Furthermore, we discussed three application scenarios and presented appropriate aggregation rule sets.

[3]Diadem Firewall is partly funded by the European Commission (FP6 IST-2002-002154). Homepage: http://www.diadem-firewall.org

References

[1] K. G. Anagnostakis, S. Ioannidis, S. Miltchev, J. Ioannidis, M. Greenwald, and J. M. Smith. Efficient Packet Monitoring for Network Management. In *8th IEEE Network Operations and Management Symposium (NOMS)*, Florence, Italy, Apr. 2002.

[2] B. Claise. Cisco Systems NetFlow Services Export Version 9. RFC 3954, Oct. 2004.

[3] B. Claise. IPFIX Protocol Specification. Internet-Draft, work in progress, draft-ietf-ipfix-protocol-22.txt, June 2006.

[4] L. Deri. nProbe: an Open Source NetFlow Probe for Gigabit Networks. In *TERENA Networking Conference (TNC 2003)*, Zagreb, Croatia, May 2003.

[5] F. Dressler, C. Sommer, and G. Münz. IPFIX Aggregation. Internet-Draft, work in progress, draft-dressler-ipfix-aggregation-03.txt, June 2006.

[6] B. Fortz, J. Rexford, and M. Thorup. Traffic Engineering with Traditional IP Routing Protocols. *IEEE Communications Magazine*, 40(10):118–124, Oct. 2002.

[7] F. Ghys and A. Vaaraniemi. Component-Based Charging in a Next-generation Multimedia Network. *IEEE Communications Magazine*, 41(1):99–102, Jan. 2003.

[8] N. Hohn and D. Veitch. Inverting sampled traffic. In *3rd ACM SIGCOMM Conference on Internet Measurement*, pages 222–233, Miami Beach, FL, USA, Oct. 2003.

[9] Y. Hu, D.-M. Chiu, and J. C. Lui. Adaptive Flow Aggregation - A New Solution for Robust Flow Monitoring under Security Attacks. In *IEEE/IFIP Network Operations and Management Symposium (IEEE/IFIP NOMS 2006)*, pages 424–435, Vancouver, Canada, Apr. 2006.

[10] R. Kemmerer and G. Vigna. Intrusion Detection: A Brief History and Overview. *IEEE Computer - Special Issue on Security and Privacy*, pages 27–30, Apr. 2002.

[11] R. T. Lampert, C. Sommer, G. Münz, and F. Dressler. Vermont - A Versatile Monitoring Toolkit for IPFIX and PSAMP. In *IEEE/IST Workshop on Monitoring, Attack Detection and Mitigation (MonAM 2006)*, Tuebingen, Germany, Sept. 2006.

[12] T.-H. Lee, W.-K. Wu, and T.-Y. W. Huang. Scalable Packet Digesting Schemes for IP Traceback. In *IEEE International Conference on Communications*, Paris, France, June 2004.

[13] J. Li, M. Sung, J. Xu, and L. Li. Large-Scale IP Traceback in High-Speed Internet: Practical Techniques and Theoretical Foundation. In *IEEE Symposium on Security and Privacy*, Berkeley, CA, USA, May 2004.

[14] J. Mirkovic and P. Reiher. A Taxonomy of DDoS Attack and DDoS Defense Mechanisms. *ACM SIGCOMM Computer Communication Review*, 34(2):39–53, Apr. 2004.

[15] J. Quittek, S. Bryant, B. Claise, and J. Meyer. Information Model for IP Flow Information Export. Internet-Draft, work in progress, draft-ietf-ipfix-info-12.txt, June 2006.

[16] J. Quittek, T. Zseby, B. Claise, and S. Zander. Requirements for IP Flow Information Export (IPFIX). RFC 3917, Oct. 2004.

[17] G. Sadasivan, N. Brownlee, B. Claise, and J. Quittek. Architecture for IP Flow Information Export. Internet-Draft, work in progress, draft-ietf-ipfix-architecture-11.txt, June 2006.

[18] C. Schmoll, J. Quittek, S. Tartarelli, S. Niccolini, T. Dietz, A. Bulanza, and E. Boschi. MOME Interoperability Testing Event. Deliverable d13 of ist mome, Aug. 2005.

[19] C. Schmoll and S. Zander. NetMate - User and Developer Manual. Technical report, Feb. 2004.

[20] H. Wang, D. Zhang, and K. G. Shin. SYN-dog: Sniffing SYN Flooding Sources. In *22nd International Conference on Distributed Computing Systems (ICDCS'02)*, Vienna, Austria, July 2002.