

Seamless Dynamic Reconfiguration of Flow Meters: Requirements and Solutions

Tobias Limmer and Falko Dressler

Computer Networks and Communication Systems, University of Erlangen,
Martensstr. 3, 91058 Erlangen, Germany
{limmer,dressler}@informatik.uni-erlangen.de

Abstract In this paper, we investigate the need for seamless dynamic reconfiguration of flow meters. Flow monitoring has become a primary measurement approach for various network management and security applications. Sampling and filtering techniques are usually employed in order to cope with the increasing bandwidth in today's backbone networks. Additionally, low level analysis features can be used if CPU and memory resources are available. Obviously, the configuration of such algorithms depends on the (estimated) network load. In case of changing traffic pattern or varying demands on the flow analyzers, this configuration needs to be updated. Hereby it is essential to lose as little information, i.e. packet or flow data, as possible. We contribute to this domain by presenting an architecture for seamless reconfiguration without information loss, which we integrated into the monitoring toolkit Vermont. Additionally, we integrated support for situation awareness using module specific resource sensors. In a number of experiments, we evaluated the performance of Vermont and similar flow monitors.

1 Introduction

Flow monitoring is becoming a dominant metering technique in professionally managed networks. Mainly, there are two reasons for network providers to measure their network traffic. First, the amount of transferred data is monitored for accounting purposes. All monitored packets are assigned to single IP addresses or specific subnets and aggregated to customer-related records, which contain information about the IP traffic. Secondly, the area of security also makes use of network monitoring: intrusion detection, attack detection, scan detection and forensic analysis are just a few application domains [1].

Usually, flow monitoring is performed using statically deployed monitors with predefined configuration settings. This procedure has two drawbacks. First, the configuration of sampling algorithms and filters must be defined for a medium load scenario. Thus, resources are wasted in case of low network load (it would be possible to inspect all packets instead of a subset) and in case of extreme load, the monitor will not be able to process all packets, which leads to non-deterministic packet drops. In order to adapt it to changing network conditions, reconfiguration of the monitor, e.g. modifying the packet sampling rate, would be

needed. Secondly, in the security context, often high speed networks are scanned for anomalies, as processing of more detailed network data would be computationally too expensive. If anomalies are detected, usually single hosts or subnetworks are involved. For more information about the cause of the anomaly, a detailed analysis of the subnetwork’s traffic is required. Again, dynamic reconfiguration of the network sensors to supply detailed data about the affected subnetwork is an adequate solution.

We analyzed the capabilities of state-of-the-art flow meters for their capabilities to provide such reconfiguration. We discovered that there is no direct way for such parameter updates nor for adding new functionality during runtime. Usually, the monitor needs to be stopped, (re-)configured, and started again. This leads to information loss during the reconfiguration process.

Based on all these observations, we developed a novel architecture for seamless dynamic reconfiguration and integrated it into our monitoring toolkit Vermont [2].¹ In order to achieve the desired behavior, we extended the modular structure of Vermont to add internal queues between all modules. Using these queues, it is possible to change the modules’ organization, to add and to remove modules without stopping all monitoring and processing activities. Without loss of generality, in this paper, we concentrate on flow monitoring for network security purposes because the requirements are covering all problem domains of other applications as well.

In this paper, we first analyze the characteristics of flow monitoring with respect to adaptive configuration of parameters, e.g. the sampling rate, or the complete update of the monitoring functionality (Section 2). We not only integrated our developed reconfiguration architecture in Vermont (Section 3.2) but also added means for situation awareness based on integrated sensors that provide information about the current resource utilization of Vermont modules (Section 3.3). In several experiments, we analyzed the performance of Vermont in comparison with other flow meters. The main focus was the monitoring performance during reconfiguration tasks (Section 4).

2 Overview and Problem Statement

2.1 Flow Monitoring

Flows are sets of IP packets sharing common properties. A flow record contains information about a specific flow. In most applications, a typical configuration would be using the IP 5-tuple $\langle source\ IP, dest\ IP, source\ port, dest\ port, protocol \rangle$ as flow keys, i.e. attributes describing the flow. Furthermore, relevant statistical data can be added such as the flow start and end times or the number of bytes of all packets belonging to the flow. Several protocols are available to efficiently transfer flow records. Most of the state-of-the-art flow meters support either Netflow.v9 or Internet Protocol Flow Information Export (IPFIX). The latter one was standardized by the IETF in RFC 5101 [3]. Both protocols support

¹ <http://vermont.berlios.de/> and <http://www.history-project.de/>

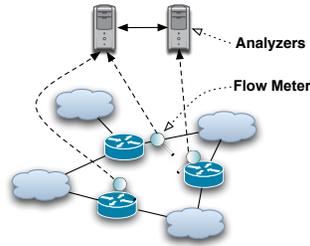


Figure 1. Example of a distributed monitoring system for integrated attack detection

variable configurations: so called template records are transmitted that describe the structure and content of flow records. Flow records are exported regularly according to predefined timeouts. The active timeout describes the maximum time a flow record is kept in cache and the passive timeout is used if no more packets are received for the particular flow.

2.2 Requirements in the Security Domain

Attack detection methodologies require input data with different levels of detail. On the one hand, there are systems available that operate on raw packet data including full payload for a detailed analysis of the monitored traffic. On the other hand, systems process summarized data that contains e.g. aggregated information about traffic volumes for subnets.

We assume an attack detection system that comprises flow monitors that are directly attached to the observed network links, and analyzers that process the collected flow data in order to detect attacks or intrusions. The flow data may have different levels of detail. Figure 1 shows an overview of the architecture. Flow monitors can be chained to support flow aggregation. Multiple hierarchically structured flow aggregators are also a topic of the IPFIX working group [4]. The depicted analyzers may execute different algorithms ranging from attack detection to application identification. Besides simple anomaly detection methods like top-N lists, more intelligent traffic summaries [5] or horizontal portscan detection [6] could be supported. Many application identification methods are also based on IP header data as available in flows [7–9].

2.3 Challenges

Today’s backbone networks maintain high data rates, where it is only possible to get sampled flow statistics for further analysis. Due to the nature of less accurate information, the detection of security incidents becomes more difficult. Similarly, detected anomalies require further inspection of suspicious hosts and connections. The monitoring infrastructure needs to provide more detailed information of potentially malicious traffic. To achieve this, the configuration of the flow meters needs to be temporarily adapted. Furthermore, even normal traffic

behavior changes over time. This may influence the load of different modules in the detection system. The idealistic goal is to keep the monitoring system always as effective as possible, thus, there is demand to update the parameters of the flow meters according to the current traffic conditions.

Rajab et al. [10] demonstrated that distributed monitoring decreases time between the outbreak of worms and their detection. This suggests a collaboration between different network operators. As sensitive information between those entities must not be transferred, information exchange should be held at a possible minimum. The best way is to reduce the level of detail of transferred information at the cost of less accurate attack detection. Only in specific cases, the level of detail may be increased in a well-controlled manner. Finally, direct attacks on monitoring infrastructures may cause the equipment to fail and create holes in accounting and performance logs, or security-related incidents may not be detected. Current networking equipment, especially in the area of flow monitoring, offers only limited capabilities for reconfiguration if it detects overload and its monitoring functionality may be impaired.

All these scenarios suggest a dynamic solution for flow aggregation: a network of sensors tries to deliver exactly the data that is needed for efficient traffic analysis by the detection algorithms.

3 Dynamic Reconfiguration

In this section, we will describe the basic concepts of our monitoring toolkit Vermont. We continue with the developed architecture for seamless reconfiguration, which has been specifically designed for Vermont. However, the basic principles can easily be adapted for other flow meters as well. Finally, we briefly cover the capabilities of Vermont to detect the resource consumption of currently running modules to support situation-aware reconfiguration steps.

3.1 Vermont

Vermont is an open-source monitoring toolkit capable of processing Netflow.v9 and IPFIX conforming flow data. It has been developed in collaboration with the University of Tübingen. The application runs on Linux and derivatives of BSD. It can receive and process raw packets via Packet Capturing (PCAP) (up to 1 GBit/s) as well as IPFIX/Netflow.v9 flow data. Supported data formats for export are IPFIX, Packet Sampling (PSAMP), and Intrusion Detection Message Exchange Format (IDMEF). The following modules are available:

- *Importers* capture raw data via PCAP, receive Netflow.v9 and IPFIX flow data via UDP and Stream Control Transmission Protocol (SCTP)
- *Samplers and filters* provide sampling algorithms and packet filter definitions
- *Exporters* export data using IPFIX, PSAMP, or IDMEF
- *Aggregators* aggregate incoming data according to customizable rules
- *Analyzers* detect anomalies in flows and output IDMEF events

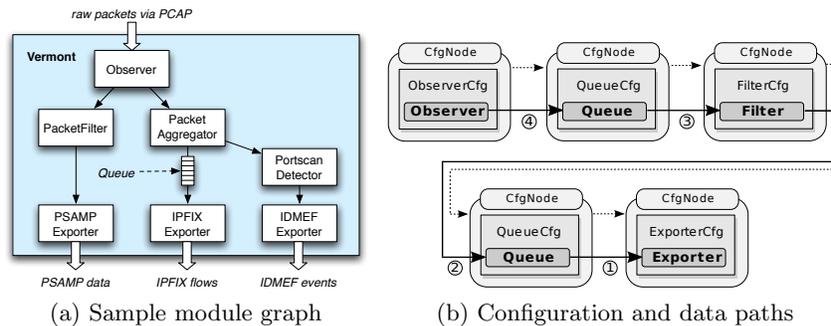


Figure 2. Vermont module configuration

Modules can be linked in almost any combination: only the input and output data type of linked modules need to be compatible. Modules may also have more than one succeeding and preceding module. Figure 2a shows an example for an arrangement of several modules. In this configuration, Vermont captures packets using PCAP, filters these packets and exports the selected PSAMP records. A second branch aggregates flows, which, in turn, are exported using IPFIX and analyzed in a portscan detector, respectively.

3.2 Reconfiguration

A special feature of Vermont is its support for dynamic reconfiguration of the module structure. Linked modules in Vermont correspond to a directed acyclic graph and operate independently from each other.

The idea is to support updates of the configuration file and to reconfigure Vermont accordingly at runtime. For this reconfiguration, Vermont computes the differences between the old and new configuration. Unique IDs are used to identify the modules. Vermont always tries to reuse existing modules in order to allow keeping state information and to speed up the reconfiguration process. If the configuration of an existing module has been changed, Vermont tries to reuse it and applies updates on-the-fly. If it is not possible to reuse a module, a new one is created. Examples are aggregator modules: for aggregation configurations, no on-the-fly reconfiguration is allowed because the used hash tables need to be rebuilt. Thus, all stored flows need to be exported and sent to the subsequent module in the module graph. This ensures as little flow data loss as possible. This process is repeated for each module until instances for all new modules are created. Modules are reconnected according to the new configuration and started in reverse topological order as depicted by the numbers in Figure 2b.

If modules do not have any asynchronous tasks to perform, they may be executed synchronously using a single thread. If, on the other hand, Vermont runs on a multicore machine, the software can be configured to use multiple threads, at most one per module. Asynchronous execution of modules causes lags in the processing time, so Vermont may use queues between modules to

compensate this problem. The queues can be fully customized, but usually FIFO scheduling with a configurable size is used. The queues block if the maximum size is reached.

Figure 2b shows a configuration consisting of three modules that are connected by queues. Shown are the configuration paths (dashed lines) that link all the modules in the module graph and the data paths (thick lines) that depict the data flow between the modules.

The development of the reconfiguration process focused on minimizing the time during which data processing is stopped. It is technically not feasible to provide completely uninterrupted processing because the dependencies between the modules need to be considered. Especially, it is not possible to reconfigure the module graph without stopping the modules that need to be re-ordered in the graph. We minimized the module's outage by preparing new modules before the processing is stopped. Additionally, the shutdown of old modules is performed after the new configuration is completed and started.

We achieved downtimes smaller than 5 ms using this method. On a link transferring 1 GBit/s, this timeout could result in a data loss of about 650 KByte. Vermont is able to buffer this data during the reconfiguration process using the memory-mapped PCAP library.² For our tests, this buffer was set to 64 MByte.

3.3 Situation Awareness

Dynamic adaptation to current traffic data rates and corresponding load on flow meters does not only depend on seamless reconfiguration, but also on the ability to identify and, in the best case, anticipate bottlenecks in the monitoring hierarchy. We implemented sensors inside Vermont to retrieve information about the current load of the system. Each module offers standard measurement values like CPU utilization and memory requirements. Additionally, module-specific data is monitored, e.g. the current packet rate or the queue size. This information is an essential requirement for algorithms that try to balance load among multiple flow aggregation nodes. Based on the data coming from the sensors, it is possible to move a task to a different system that still has unused capacities.

Figure 3 shows example statistics from the aggregator's hash table that were collected over one day: the black line shows the total number of entries inside the hash table, the blue line shows the number of entries that shared a single bucket with other entries inside the hash table. Multi-entry buckets considerably slow down the lookup of entries in a hash table, as they are implemented as linked lists. In our example, the hash table offered a total of 256 Kbuckets, but at the time of 800 min a DDoS attack occurred on the monitored link and the number of entries exceeded the hash table's capacity by far. This is a typical case for a DoS attack against the flow meter and should be evaded by monitoring the module load and adequate reconfiguration.

² <http://public.lanl.gov/cpw/>

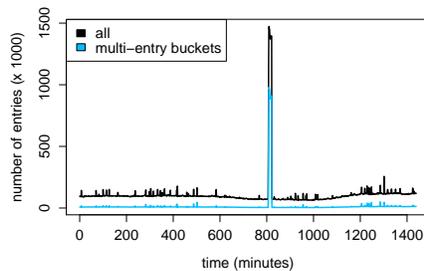


Figure 3. Hash table size

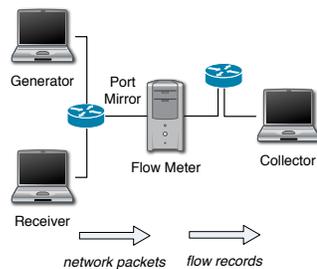


Figure 4. Testbed setup

4 Performance Measurements

4.1 Test Setup

In this section, we analyze the reconfiguration performance of Vermont and compare it to the performance of typical flow meters. We set up a testbed for these experiments in which the different flow monitors are tested with artificial traffic as well as with real traffic. Figure 4 shows the structure of the test setup. We used a dedicated PC for generating traffic, which, in turn, was forwarded to the system under test. The forwarding was performed using a mirror port on a layer-2 switch. The flow meters exported their data on a separate network interface to a flow collector, which logged the received flows for later analysis. Care was taken to deactivate all unneeded functions of the systems, including the switches, for more deterministic results.

For performance comparison, we used a Cisco Catalyst 6500 router running IOS firmware version R12.2 SXF. It supports basic flow aggregation for routed traffic using a “NetFlow cache”, which exports Netflow.v5/v9 data. More flexible configuration options are offered by the feature called “NetFlow aggregation”. It uses an additional NetFlow cache table in the switch called “Aggregation cache” which has aggregated flow statistics of the monitored traffic. Different aggregation schemes like “source prefix” (aggregation according to source IP address), “destination prefix” (destination IP address), “prefix-port” (both IP addresses and ports), “prefix” (both IP addresses) and more are supported, and for each aggregation scheme several collectors of the flow data may be specified. The parameters for the timeout of active and inactive flows are specified in minutes and seconds, respectively. So the minimum timeout for active flows is 1 min and for inactive flows 1 s.

Another popular flow meter is nProbe (Netflow probe), an open source Netflow.v5/v9 and IPFIX probe. Similar to Vermont’s basic functionality, it captures packets on an Ethernet network, aggregates the packets to flows, and exports them. The main focus of nProbe is its efficiency to support fast flow aggregation in software. Attributes of exported flows are fully configurable. According to the documentation, aggregation is performed identically to Vermont: incoming packets are inserted into a hash table, which in turn is regularly checked for

flows to be exported. Those timeouts are also configurable on the command-line. On-the-fly reconfiguration is not supported by nProbe.

4.2 Experiment Description

In *configuration phase 1*, we aggregated flows according to the IP 5-tuple. In *configuration phase 2*, the flow meters were instructed to omit source and destination ports in the flows, thus producing less flows containing more packets. These two configuration settings are similar to the aggregation schemes “prefix-port” and “prefix” provided by the Cisco router and ensure that different rules must be used for aggregation. Parameters controlling the timeout for inactive and active flows were set to 10 s and 60 s, respectively. Both nProbe and Vermont were configured to use a scan / export interval of 10 s for the internal hash table. The testing process involved the following steps:

1. start collector and flow meter using *configuration setting 1*
2. start traffic generator
3. reconfigure flow meter for *configuration setting 2*
4. wait until flow meter finished sending flows to collector

Reconfiguration of Vermont was performed as described in Section 3.2. We ensured that the other tools were reconfigured in as little time as possible: nProbe does not explicitly support dynamic reconfiguration, so we executed nProbe using the settings of *configuration phase 1*, terminated the process using the SIGINT signal at reconfiguration time, and immediately restarted the process. The Cisco router was reconfigured using an already established telnet connection: old flow aggregator settings were shut down and then the new scheme was activated by transmitting the corresponding commands to the machine.

We generated two different types of traffic for the test: artificially generated traffic, where all packets shared almost identical properties, and real-world traffic. As our goal was to test traffic losses caused by reconfiguration, we only generated low data rates, so that none of the flow meters got overloaded. For generating artificial traffic, we used the Network Packet Generator (Npag) tool. It produced TCP packets with constant source and destination IP, constant source port, no payload and the destination port was uniformly distributed in a specified range. Additionally, we used tcpdump and tcpreplay³ to record traffic from a LAN and to replay the traffic to the flow meter, respectively, so that the flow meters processed almost identical traffic. The network dump had a length of 71 s. It counted 150 608 packets in total and had an average data rate of 11.69 MBit/s or 2111 packets/s. The traffic was generated for a total of 60 s. Reconfiguration was performed 30 s after the traffic generator was started.

4.3 Further Issues

If the flow meter is reconfigured or stopped, all cached flow information needs to be exported. In the best case, already processed data records are exported

³ <http://www.tcpdump.org/> and <http://tcpreplay.synfin.net/>

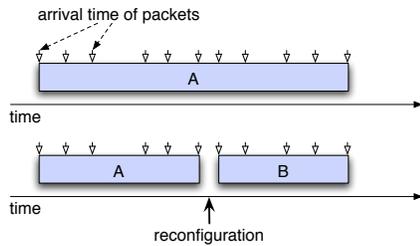


Figure 5. Flow splitting problem

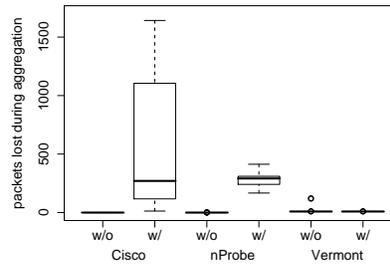


Figure 6. Packet loss for `tcpdump` traffic

immediately after reconfiguration. Simultaneously, newly arriving packets are processed using the new configuration and exported with a new flow description. This ensures that all packets are reported using the configuration that was valid at reception time and minimizes time until flow data is sent to the collector. A side-effect of this immediate configuration switching is that flows are split into two parts. An example is depicted in Figure 5. On the upper part, a flow is shown matching configuration A. If reconfiguration is performed during the lifetime of the flow (lower part), the flow is being split. This effect is unavoidable, as otherwise the aggregator would not follow its configuration semantics. Depending on the configuration, it will not be possible to join both parts at the collector.

4.4 Results and Discussion

In the following, we present selected results of our experiments. Primarily, we focus on the packet loss caused by reconfiguration. Furthermore, we investigate the flow export times of the different flow meters with and without reconfiguration.

Packet Loss We compared the number of packets contained in all flow records exported by the flow meter with the number of packets sent by the traffic generator. The results are depicted in Figure 6 using boxplots: a box is drawn from the first quartile to the third quartile, and the median is marked with a thick line. Additional whiskers extend from the edges of the box towards the minimum and maximum of the data set. For this experiment, we used `tcpreplay` for testing the system with real-world traffic. For all the systems, two box plots are displayed without and with reconfiguration. We performed 30 test runs for statistical validity and to identify outliers. Without reconfiguration, all flow meters performed very well with almost no packet loss. With reconfiguration, only Vermont shows the loss rate close to zero, whereas Cisco and nProbe kept much higher loss rates – an average downtime of 130 ms was observed. nProbe needs to be completely restarted for reconfiguration. Thus, there is a short time period in which no packets can be recorded. This explains nProbe’s loss of roughly 300 packets. The Cisco router performed much worse losing about 550 packets on average. The high variance cannot be explained without deeper insights into the internal flow processing.

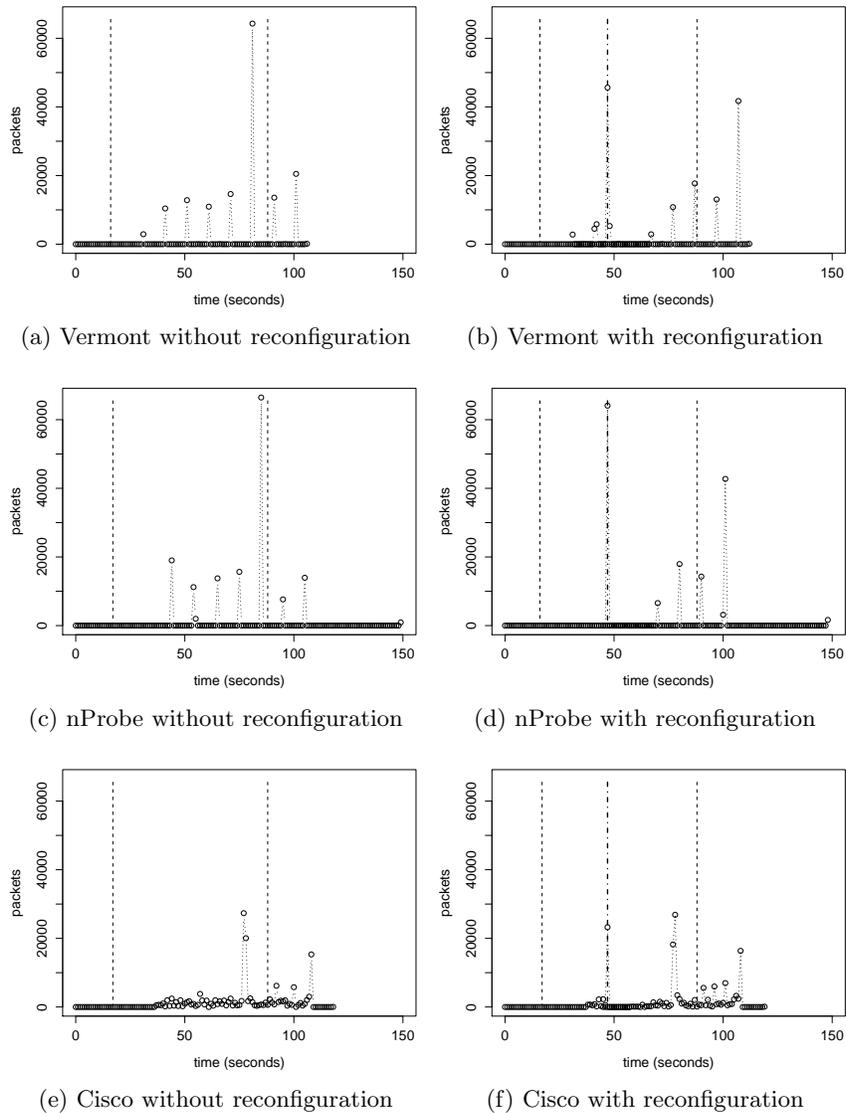


Figure 7. Characteristics of the network sensors' flow export

Flow Export Times All generated flow records were logged by the collector including a time stamp recording the time of reception. This way, we were able to relate the start and end times of the flow, i.e. when it was monitored by the flow meter, to the time when the flow record was exported. Figures 7a–7f show such relations: the start and end times of traffic generation are marked by the dashed vertical lines. In all figures with reconfiguration, a vertical bar at 30s

after start time indicates the switch from *configuration phase 1* to *configuration phase 2*. The dotted lines with markers show the amount of packets contained in flows that were sent to the collector, displayed at the time of reception at the collector. All figures show the captured, real-world traffic replayed by `tcpreplay`. The same tests were performed with artificial traffic and confirmed the presented results.

For Vermont, the results without reconfiguration are shown in Figure 7a. It can clearly be seen that Vermont was configured to check its aggregation hash table every 10s and then immediately export the expired flows to the collector. The high spike at 80s is caused by the active timeout of 60s, thus, all active flows are contained in the corresponding flow records. nProbe uses almost the same aggregation technique as Vermont. Thus, Figure 7c does not show many differences. Interesting is the small spike at 150s, where the last flow records were exported. According to the configuration, this export comes much too late, as no flows should be active after stopping the traffic generator. Passive flows were to be cached for 10s and at most 1s was waited until records were exported. Figure 7e shows that the Cisco router uses a different aggregation scheme compared to the software-based flow meters. Flows were not exported in regular intervals, but continuously after their expiry. First flows were exported 10s after traffic generation started. The spike at 80s corresponds to the configuration settings (see Vermont and nProbe).

For reconfiguration, Vermont shows the expected outcome in Figure 7b. All flows are exported at reconfiguration time and the succeeding exports show regular behavior. The same applies to nProbe, including the aforementioned spike at 150s that is not conform to the configuration (Figure 7d). The Cisco router shows an interesting behavior depicted in Figure 7f. There is a spike in the graph that shows the expected peak at reconfiguration time, but there are less packets exported compared to Vermont or nProbe. Additionally, there is a spike 60s after start that seems to be roughly as high as the spike in Figure 7e at the same time. These spikes are caused by exported active flows and should not show the same values as all cached data should have been exported at reconfiguration. We concluded, that the Cisco router does only export passive flows at reconfiguration time and continues caching all active flows that are exported at the same time as they would have been without reconfiguration. This means, that flows monitored during *configuration phase 1* are exported using the settings of *configuration phase 2*. This behavior is weird because exported data can not be associated to the correct phase.

5 Conclusion

We demonstrated the need for seamless dynamic reconfiguration of flow meters for various application fields. Based on this motivation, we presented a new solution for this issue: the flow monitor Vermont, which offers a modular structure that is optimized to guarantee short reconfiguration times and offers in-depth self-monitoring capabilities for situation-aware reconfiguration decisions. In our

performance tests, which included the software-based flow aggregator nProbe and a high-end Cisco router, we showed that currently available flow meters do not offer the needed reconfiguration performance. Typically, an average downtime of 130 ms was observed in which all incoming packets were dropped and not reported to the flow collector. For complex hierarchical analysis systems, reconfiguration procedures should be seamless without noticeable data loss. The implemented architecture showed its advantages during this test. Due to its modular, data flow oriented structure, the overhead of a reconfiguration process can be kept at a minimum – no packet loss has been observed.

Acknowledgments

The authors would like to thank Christoph Sommer for his work on Vermont and his standardization efforts, and Peter Baumann for the implementation of the basic reconfiguration concept within Vermont.

References

1. Carle, G., Dressler, F., Kemmerer, R.A., König, H., Kruegel, C., Laskov, P.: Manifesto - Perspectives Workshop: Network Attack Detection and Defense. In: Dagstuhl Perspectives Workshop 08102 - Network Attack Detection and Defense 2008, Schloss Dagstuhl, Wadern, Germany (March 2008)
2. Lampert, R.T., Sommer, C., Münz, G., Dressler, F.: Vermont - A Versatile Monitoring Toolkit Using IPFIX/PSAMP. In: IEEE/IST Workshop on Monitoring, Attack Detection and Mitigation (MonAM 2006), Tübingen, Germany, IEEE (September 2006) 62–65
3. Claise, B.: Specification of the IP Flow Information Export (IPFIX) Protocol for the Exchange of IP Traffic Flow Information. RFC 5101, IETF (January 2008)
4. Kobayashi, A., Nishida, H., Sommer, C., Dressler, F., Stephan, E., Claise, B.: IPFIX Mediation: Problem Statement. Internet-Draft (work in progress) draft-ietf-ipfix-mediators-problem-statement-00.txt, IETF (May 2008)
5. Estan, C., Savage, S., Varghese, G.: Automatically Inferring Patterns of Resource Consumption in Network Traffic. In: ACM SIGCOMM 2003, Karlsruhe, Germany, ACM (August 2003) 137–148
6. Jung, J., Paxson, V., Berger, A.W., lakrishnan, H.B.: Fast Portscan Detection Using Sequential Hypothesis Testing. In: IEEE Symposium on Security and Privacy, Berkeley/Oakland, CA (May 2004)
7. Bernaille, L., Teixeira, R.: Early Application Identification. In: 2nd International Conference On Emerging Networking Experiments And Technologies (CoNext 2006), Lisboa, Portugal (December 2006)
8. Crotti, M., Dusi, M., Gringoli, F., Salgarelli, L.: Traffic Classification Through Simple Statistical Fingerprinting. ACM Computer Communication Review (CCR) **37**(1) (January 2007) 5–16
9. Wagner, A., Dübendorfer, T., Hämmerle, L., Plattner, B.: Identifying P2P Heavy-Hitters from Network-Flow Data. In: 2nd CERT Workshop on Flow Analysis (FloCon 2005), Pittsburgh, Pennsylvania (September 2005)
10. Rajab, M.A., Monroe, F., Terzis, A.: On the Effectiveness of Distributed Worm Monitoring. In: 14th USENIX Security Symposium, Baltimore, MD (July 2005)