A distributed multi-purpose IP flow monitor

Claudio Mazzariello, Francesco Oliviero
Dipartimento di Informatica e Sistemistica — Università degli Studi di Napoli "Federico II"
Via Claudio 21, 80125, Napoli, Italy
E-mail: {`cmazzari,folivier`}`@unina.it`

Salvatore D'Antonio, Dario Salvi
Lab. ITeM - Consorzio Interuniversitario Nazionale per l'Informatica - CINI
Via Diocleziano 328, 80124, Napoli, Italy
E-mail: {`salvatore.dantonio,dsalvi`}`@napoli.consorzio-cini.it`

Abstract

Traffic monitoring is a research field whose results can be exploited for several purposes, such as network resource management, security and accounting. An effective monitor needs to be capable of analyzing the traffic flowing through the monitored network by losing as few packets as possible since packet loss may result in a non accurate measurement of the required metrics. Such a monitor captures the packets from the network, associates each packet to a flow by evaluating its characteristics, performs some flow measurements, and exports the results of data analysis. In high speed networks such tasks might be hard to accomplish in an efficient way, as the number of analyzed flows is very high. For this reason, we decided to design and implement a distributed monitoring system comprising several components each responsible for a different task. Such a distributed approach helps overcome the problem of an overloaded monitoring system. Furthermore, distributed systems need an appropriate protocol, that defines the kind as well as the sequence of messages exchanged between system components. In this paper we present both the monitoring architecture and the corresponding management protocol. Finally, in order for the monitoring system to support different kinds of applications, we developed an open framework allowing a user to define a customized set of metrics.

Keywords: traffic measurements, IP flow monitoring, Last Recently Used caching, traffic profiling, intrusion detection

1 Introduction

This paper aims to introduce a distributed, highly configurable, multipurpose architecture for IP flow monitoring. An *IP flow* is defined as a sequence of packets sharing some properties characterising the packet headers. A large number of applications benefit from the availability of accurate information about traffic flows.

One first example is *traffic profiling*, which consists of the development of analytical models used to predict the traffic behavior.

The traffic profiling activity cannot be separated from traffic measurement due to the fact that any modeling requires the measurement of some metrics both as input to the model and for the validation of the model itself. The modeling done at the IP flow level is a recent, challenging issue [Claffy et al., 1995].

A further example of application relying on measurements is the *intrusion detection* in IP networks. An intrusion detection system can be made of two modules: a monitoring system that measures a set of metrics related to the observed flows, and a detection engine that analyzes the metrics in order to detect a "malicious" behavior. A large number of intrusion detection systems are based on the analysis of packet-level or TCP-connection-level data [Bace, 2000]. Recently a new IP flow level analysis has been introduced in order to detect malicious network activities [Barford and Plonka, 2001].

Therefore, different applications rely on the measurement process. Unfortunately different applications use different kinds of metrics; for example both cited applications need the definition as well as the computation of specific metrics. For this reason we propose an open architecture allowing to implement a set of algorithms to calculate metrics by means of an application program interface (API).

Another challenging task is the development of a system for flow monitoring in high speed networks. Due to the high throughput characterizing such networks, a first requirement is imposed to the monitoring system: data have to be quickly and effectively collected in order to operate in scenarios with a large number of simultaneous flows (up to millions on a OC192 link) and a short packet inter arrival time (few nanoseconds on a OC192 link). If the measuring system is not able to manage a large number of flows or short inter arrival times, a loss of flow data or packets may occur. In both cases the applications miss vital information. For this reason we developed a distributed architecture that can measure a large number of flows during relatively short times.

Following an analysis of related work in 2, in section 3 we describe the architecture of our monitoring system. In order to develop a reliable and robust system, we have defined a proper protocol to manage the communication among the different modules of the distributed architecture. The protocol definition is in section 4. In section 5 we discuss some scalability issues. Finally the section 6 provides some concluding remarks, together with information concerning our future work.

2 Related Work

There is a rich literature about flow monitoring. Here we want to focus on the most recently proposed architectures, in particular those presenting some level of distribution.

One important architecture has been developed by the IETF RTFM working group (Real Time Traffic Flow Measurement) [Brownlee et al., 1997]. The architecture is composed of the following modules:

1. A *manager* which configures and controls the other two modules on the basis of the requirements.

2. A *meter*, which is the core module of the system. It analyzes the traffic passing through one or more network interfaces and aggregates it in flows.

3. A *meter reader* which collects the data read by the meter and sends it to the applications.

Another interesting IETF working group is IPFIX (IP Flow Information eXport) [Sadasivan et al., 2005]. This group was born in 2003 and its goal is to define a common architecture and protocol to let different monitoring applications communicate to each other. The proposed architecture consists of two modules: the *IPFIX device* and the *collector*.

The IPFIX device captures packets and measures the flow data. It is made of four components:

1. The *observation domain* which is a set of points of observation. A point of observation is a location on the network where it is possible to observe IP packets.

2. The *metering process* is the unit which captures packets from an observation point and generates the *flow records*, a data structure where all the statistics related to the flow are stored. Each observation point must be associated to, at least, one metering process. The metering process must be able to capture packets, sample packets, generate the flow key for each packet, update the flow records.

3. The *flow recording process* stores all the flow records sent by one or more metering processes. Its task is to keep in memory all the flow records related to active flows, i.e. flows which are still sending packets.

4. The *exporting process* must send all the measured data to the collector. The used protocol is defined by the working group.

The collector is the module which collects all the information sent by the exporting processes, stores the data, and sends it to the applications.

Finally, we cite three works where real time flow measurement is done by means of a distributed tool.

In [Mao et al., 2001] the authors present a distributed real time system for web traffic analysis. The architecture comprises three modules:

1. A *network dispatcher* which dispatches the captured packets among multiple analysis nodes. The main task of this module is to split the set of traffic flows in subsets in order that they are approximatively the same average load for the analysis node.

2. An *analysis node* which performs the traffic analysis. In this paper measurements concern web traffic, but the architecture could be generalized with little modifications.

3. A *storage center* which collects all the measured data sent by the analysis nodes.

In [Kitatsuji and Yamazaki, 2004] the authors propose a distributed tool for IP flow measurement based on a general definition of flow. The framework is composed of the following entities:

1. A *distribution device* which forwards packets to multiple capturing devices.

2. A *capturing device* which identifies packets that correspond to flow definitions and computes statistics. Statistics are periodically sent to a manager device which returns flow definition updates.

3. A *manager device* which stores data received from multiple capturing devices, obtains flow definitions from the user interface devices and sends them to the capturing devices.

4. A *user interface device* which provides users with an interface.

In [Han et al., 2002] the authors present a monitoring system[1] for flow based analysis in high speed networks. The architecture is made of five, fully distributed, modules:

1. A *splitting device* which sends the packets to the packet capturer.

2. A *packet capturer* which captures the packets sent by the splitting device, maintains the minimum header information for each packet, adds a time stamp, and sends the data to the next module.

3. A *flow generator* which stores the flow data in its memory for processing it. Data related to each expired flow is then exported to the next module.

4. A *flow store* is a database where all the recent data is stored. Since many queries, insertions from flow generators and selections from the next modules can be executed, it is implemented as a distributed system as well. All the read data is discarded in order to store only a minimum amount of data.

5. A *traffic analyzer* which makes queries on the stored data according to the analysis purpose.

With respect to the cited architectures, we can remark that they do not allow to define metrics in a flexible way. Since different applications could be interested in different metrics, enabling the metric customization represents a valuable functionality. This is one of the main contributions of our work.

3 The monitoring architecture

The system we want to present is in charge of capturing packets from the network, associating them to a flow, by the means of a customizable flow definition, and updating a record containing flow-related metrics. Measured data is, therefore, collected in order to make it available to the applications. The system architecture comprises a module that stores all the flow records related to *living* flows, where a living flow is a flow which is still receiving packets before a timeout occurs. The main issue concerning this task is represented by the fact that the number of living flows is very high (up to millions) and the packet inter-arrival time is very short on high speed links. This implies that the time interval spent to search for the record associated with a captured packet can be longer than the packet inter arrival time, in case of a huge number of flow records. For this reason we decided to adopt a distributed approach making it possible to divide the task of keeping the records related to the living flows among multiple processes. The proposed architecture is made of the following modules:

[1] http://ngmon.postech.ac.kr/NG-MON/

1. *Meter*, which captures the packets from a network interface, or equivalently from a trace file, associates them to a flow identification number, the so-called *flow id*, and passes them to next component.

2. *Flow Cache*, which stores the metrics related to the living flows observed and updates the records. This is the most challenging component as it has to search and update the metrics within the inter arrival time. For this reason we decided to introduce more flow caches and implement each of them as a separate process. Flow records are associated to them on the basis of their flow id. The flow cache is also responsible for *exporting* all the information related to timed-out flows to a further component, i.e. the collector.

3. *Collector*, which collects the metrics related to the flows observed by all the flow caches. The collected data can be used by different applications such as traffic profiling, intrusion detection, billing, etc.

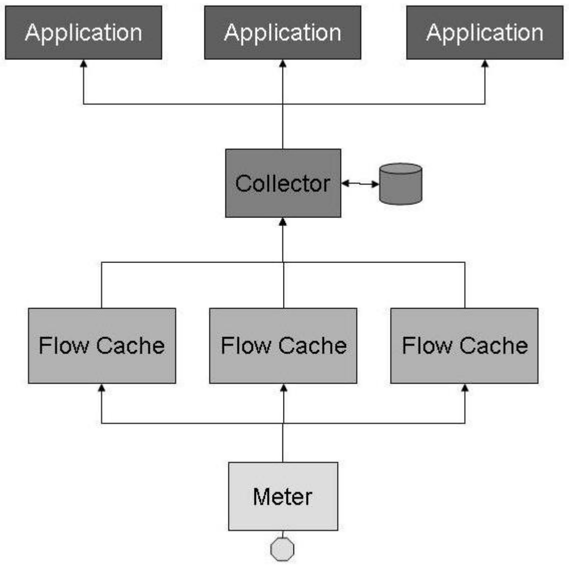

Figure 1. Architecture

Furthermore, valuable capabilities, such as on-line packet sniffing and filtering, packet analysis based on a trace file, customizable definition of a flow via a scripting language, and customizable definition of a metric via an API are provided.

A partially implemented version of the software is available on the SourceForge site at http://sourceforge.net/projects/difmon/.

3.1 Meter

The Meter performs the following tasks:

1. capturing packets from the network interface or from a trace file
2. associating a flow id to every packet
3. providing a time stamp in order to keep track of when the packet was captured
4. identifying, by means of the flow id, the flow cache which is related to the flow
5. sending the packet (together with its flow id and time stamp) to the selected flow cache

The flow id computation relies on a set of rules defined by the user by means of a standard language. Such language supports the flow definition according to the contents of the packet headers from the IP level to the application one.

The selection of the flow cache is done by means of a hash function applied to the flow id. The chosen function is the mmh which is fast enough and has good stochastic properties that enable the uniform distribution of the flow records among the various flow caches.

3.2 Flow Cache

The flow cache keeps track of living flows in order for metrics to be updated in real time. It receives the captured packet and the related flow id from the meter and, then, establishes whether a corresponding flow record already exists. If so, the flow cache updates the related metrics, otherwise a new flow record is created.

Based on the assumption that the user should be able to define specific metrics, an API is provided. This makes the system very flexible and capable to support different kinds of applications.

Moreover, the flow cache periodically sends the data related to the no-more-living flows to the collector. The definition of a living flow can be based on the introduction of a timeout or on the analisys of TCP sessions.

Some real-time applications, such as intrusion detection, may require the exporting of some still-living flows. In this case the flow cache selects living flows to be exported through a heuristic function.

The main challenge is the development of a fast and effective flow cache. In particular it is necessary to implement a suitable data structure and ordering mechanism to maintain information about living flows. We intend to apply an LRU (Least Recently Used) ordering as it is the main solution used in caching algorithms. This ordering algorithm allows addressing two issues: it provides a fast way to detect timed out flows as well as a good heuristic to select the so-called *heavy hitters* flows, where a heavy hitter flow is a high rate flow.

In fact, by scanning the LRU queue from the tail and by checking for each record whether the difference between the record's last update time and the current time exceeds the timeout, it is possible to find every timed-out flows. Therefore, if the search for heavy hitters flows is always done by scanning the LRU list from its head, one will step into heavy hitters flows with high probability. The exporting process can take advantage of this ordering mechanism simply exporting the first N records of the flow cache queue.

3.3 Collector

The collector is the module responsible for collecting the metrics of all the observed flows and sending them to the running applications. In case it receives flow records related to the no-more-living flows from the flow caches, then such records are both stored into a file and sent to the applications. In case the collector receives living flows, it passes them directly to the applications performing real time operations. As the flow cache, the collector provides an API.

4 The management protocol

In this section we present the protocol for managing interactions among the monitoring system components. The aim is to make the system robust, flexible and tolerant to every kind of faults and errors. To this purpose, when designing such a protocol we made the following assumptions:

1. the system modules run on hosts belonging to a dedicated network separated from the network to be monitored. This is for two reasons: first, the traffic generated by the monitoring system must not affect the behavior of the monitored traffic. Second, the network connecting system components should be faster than the monitored network as it has to re-transmit every captured packet plus some further information.
2. The modules may run on different machines as well as two or more modules may run on a same machine.
3. The meter is fast enough to perform packet capturing and classification within the mean packet inter arrival

time. Both the meter and the collector have to be properly designed and implemented since their behavior largely affects system performance.

4. Both the meter and the collector use well defined port numbers to send or receive signaling messages, while data transfers between system modules happen by using port numbers dinamically chosen.

In next subsections we will illustrate some use cases concerning the operation of the monitoring system.

4.1 Start And Stop

Starting the system:

We start the system by launching the meter and the collector. After that, at least one flow cache is launched.

The flow cache establishes a connection with both the meter and the collector. More precisely, the flow cache asks the meter to set up a connection, the meter answers and then waits for an ACK from the flow cache on the just established connection. Afterward, the flow cache sends a connection request to the collector, the collector replies and waits for an ACK. Finally, the flow cache sends the ACK packet to both the meter and the collector.

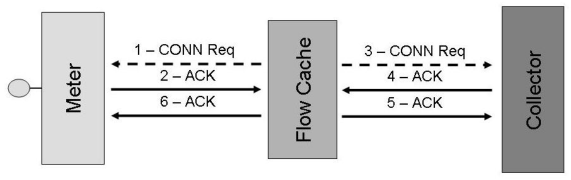

Figure 2. Start of the system

Stopping the system:

The system can be stopped by the collector (as a consequence of an application's request) or by the meter (e.g. when the trace file ends).

In the first case the collector sends an END packet to the meter which forwards it to the flow caches and waits for an ACK. As soon as the flow caches receive the END packet, they export all the stored data and then send the END packet to the collector and wait for an ACK. Once the collector has received the END packet from all the flow caches, it sends the ACKs to the flow caches and stops. The flow caches, in turn, send the ACK packet to the meter and stop. Finally, the meter stops as soon as it receives the ACK packet from all the flow caches.

In the second case the sequence of messages is the same as above except for the source of the end request.

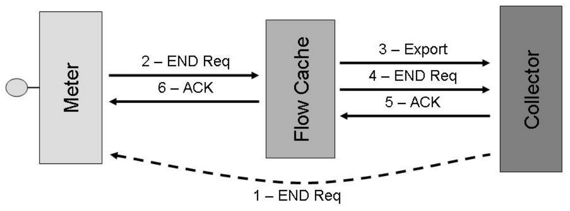

Figure 3. Stop of the system

4.2 Steady-state behavior of the protocol

Once the handshaking has ended, it is possible to define two different communication protocols between the modules.

Interaction between the meter and the flow cache:

The meter sends bulks of captured packets together with associated flow identifiers and timestamps to the flow caches. A flow control mechanism is used to manage the interaction between the meter and the flow cache. Flow control is especially useful in case packet information is retrieved from a trace file since reading from a trace file is faster than capturing packets from network.

Interaction between the flow cache and the collector:

Flow caches send bulks of measured data to the collector and wait for an ACK.

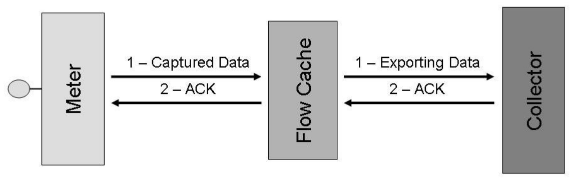

Figure 4. Steady state

4.3 Management of metrics

The metrics should be defined on the collector side, which is the only module communicating with the applications. When a flow cache sends a connection request to the collector, it replies by attaching the list of metrics to the answer.

4.4 System abort

Problems concerning one of the modules can jeopardize the operation of the whole system. In this case an ABORT message is introduced in order to stop all the modules. Three kinds of events should be considered:

1. A flow cache falls into an error.

   In this case the flow cache sends an ABORT message to both the meter and the collector and then stops. The meter and the collector forward the message to all the flow caches and each other and then will stop. The flow caches try to forward the message to both the meter and the collector and stop.

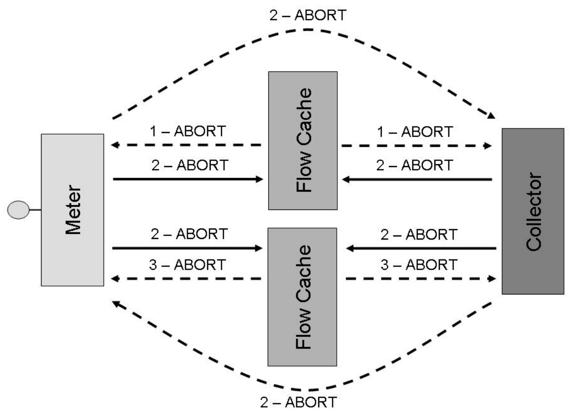

Figure 5. Abort from flow cache

2. The meter falls into an error.

   The meter sends the ABORT message to the collector and all the flow caches and, then, stops. The flow caches forward the message to both the collector and the meter and stop. The collector tries to forward the message to the meter and to all the flow caches and, then, stops.

3. The collector falls into an error.

   After sending an ABORT to the meter and to all the flow caches, the collector stops. The meter forwards the message to all the flow caches and to the collector and, then, stops. The flow caches try to forward the message to both the collector and the meter and stop.

The aborting algorithm is very simple. Once an ABORT message is received by a system module, such a message is forwarded to all the connected modules. This algorithm is not efficient as it causes a flooding of abort messages. However, since the system abort is a rare event, the proposed algorithm represents a suitable solution thanks to its simplicity and timeliness.

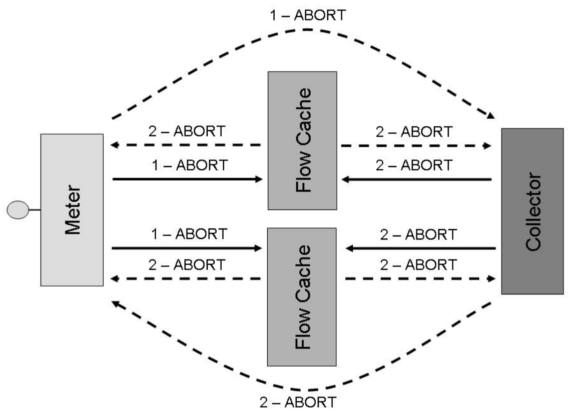

Figure 6. Abort from meter

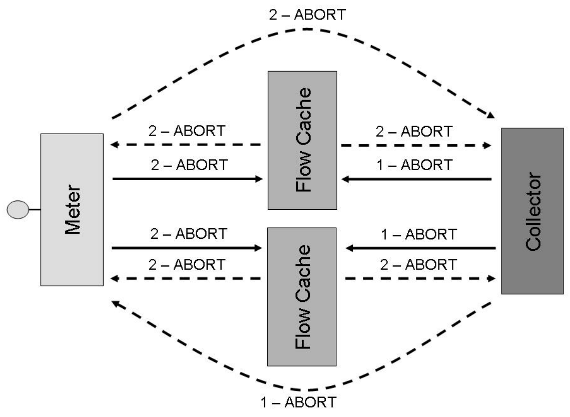

Figure 7. Abort from collector

4.5 Adding a flow cache

We want the flow caches to be able to dynamically connect and disconnect from the system, at any time during the operation.

When a flow cache starts it sends a connection request to the meter. The meter answers creating a connection and waits for an ACK. The flow cache, then, sends a connection request to the collector, the collector answers with the metrics list and waits for an ACK. At this point, the flow cache sends the ACKs to both the meter and the collector establishing the communication with them.

4.6 Deleting a flow cache

In some cases (e.g. when an error happens at flow cache side or some communication problems occur) it is useful to disconnect a flow cache from the system.

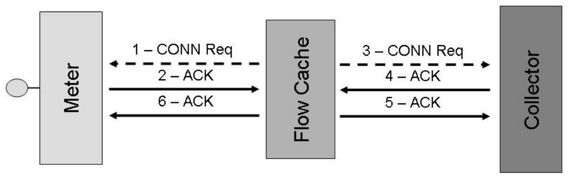

Figure 8. Adding a flow cache

When a flow cache wants to disconnect from other modules it sends a disconnection request to the meter. The meter then sends an ACK and closes the connection with that flow cache. The flow cache then performs the last exporting operation and sends a disconnection request to the collector. The collector sends an ACK and closes the connection, the flow cache can now stop.

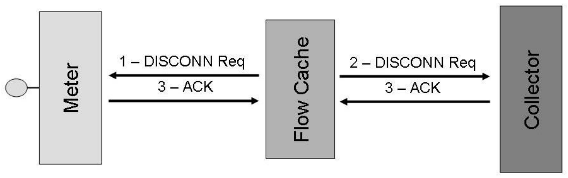

Figure 9. Deleting a flow cache

4.7 Crash of a module

1. Meter's crash:

   If the meter suddenly stops, the flow caches will get aware of this by observing that they are not receiving any more data. If the network is down, the meter does not send any data although it is active. Thus, it is necessary that the flow caches send a message to the meter to verify whether it is alive. If no data is seen at the flow cache side before a timeout occurs, the flow caches send an ALIVE request to the meter which should answer by sending an ACK to all the flow caches. If no answer is received before a timeout occurs, the flow caches abort the system.

2. Flow cache's crash

   If a flow cache crashes, both the meter and the collector can get aware of it by observing that no more data are sent from the flow cache. The meter will not receive any ACK, so it will assume that the flow

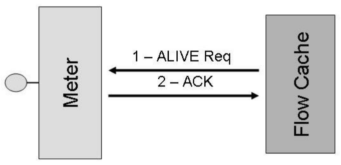

Figure 10. Meter's crash

   cache has crashed, and will disconnect it. The collector will not receive any more exporting data and, therefore, it will send an ALIVE message in order to verify whether the flow cache is still active. If the flow cache does not answer, the collector will close the connection with it.

3. Collector's crash

   If the collector crashes, the flow cache will get aware of such event by observing that no ACK is received and will abort the entire system.

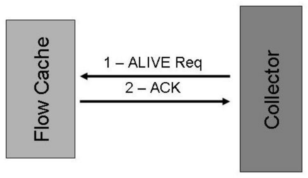

Figure 11. Collector's crash

4.8 A critical case

Let us analyze the case when a flow cache is disconnected from the meter, while keeping on being active.

It is possible that an over-loaded flow cache is not able to send the ACK packets to the meter before the timeout occurs. In this case the meter will suppose that the flow cache has crashed although it is still active. Consequently, when the flow cache ends its computations it will find itself disconnected from the meter and will send a message asking if the meter is still alive. The meter will not answer and the flow cache will assume the meter is dead and will abort the system.

It is possible to avoid this situation by observing that the flow cache which is no more connected to the meter, before sending the ALIVE request, has just sent an ACK.

In this case the flow cache should not abort the system if it does not receive any ACK from the meter, but should just disconnect from the collector and exit.

The opposite situation, where the flow cache is disconnected from the collector but still connected to the meter, should never happen because the timeout used by the collector to verify the flow cache's activity should be much longer than the meter's one. Under this hypothesis the flow cache can be disconnected from both the meter and the collector, or only from the meter.

5 Scalability Issues

We would like to discuss here about some scalability issues related to the system we have just described. There are two main problems related to the scalability, one is related to the number of flow caches, the other is related to number of sniffed networks:

5.1 Increasing the number of flow caches

What happens if the number of flow caches increases ? The main idea of this project is to implement a distributed flow cache. We have already illustrated the reasons that make the approach effective: increasing the number of flow caches helps to distribute the memory and computation load necessary during the packet inter arrival time. On the other hand, increasing the number of flow caches might represent an overload in the communication process between the meter and flow caches: new flow caches can generate new messages for the meter or the collector. From a deep analysis it is possible to observe that the sent messages do not increase as the number of flow caches varies. In fact, on the meter side, if we have one flow cache we expect one ACK message every N captured packets, but if we have M flow caches we expect M ACK messages every NxM captured packets (which means one ACK every N packets), and the number of captured packets does not depend on the number of the flow caches. Similarly, on the collector side, if we have only one flow cache, we expect one ACK message every N exported flows, while if we have M flow caches we expect M ACK messages every NxM exported flows, and the number of exported flows does not depend on the number of flow caches.

Moreover, it should be noted that there is a small traffic increase during the initial handshaking, because each flow cache has to send and receive the start messages sequence independently. This is the only difference in the network load caused by the increase in the number of flow caches.

5.2 Increasing the number of monitored networks

In this paper we have supposed that the meter captures the packets only from a single interface; but what happens if we want the meter to capture the packets from more than one interface?

Two approaches are possible: we can use only one meter to capture the packets from different network cards, or we can use more meters, each one capturing from only one interface.

In the first case we may face some scalability problems related to the fact that increasing the number of network interfaces the average packet inter arrival time decreases, overloading the meter. In this situation the meter could not be able to terminate per-packet computation before a next packet arrives, so it will lose packets.

In the second case we override scalability problems on the meter but we should allow the flow caches to connect to every active meter, needing some slight changes in the protocol.

6 Conclusions and Future Work

After analyzing functionalities provided by similar architectures, we have designed and implemented a scalable, distributed, multipurpose, reliable system for flow-based measurements. Capturing traffic from high speed networks is a challenging task due to short packet inter arrival times and huge number of concurrent flows; for this reason we propose a distributed architecture in order to achieve high performance when keeping in memory statistics related to active flows. In order to make the system reliable and robust, we defined an appropriate protocol managing the interactions between the system components. A further contribution illustrated in the paper is the implementation of an API aiming to allow users to define metrics to be measured on the flows. Such a feature is very attractive as it makes the system suitable to different contexts, such as security, traffic profiling or billing where specific metrics are of interest.

As for future work, in order to assess the feasibility of the proposed architecture a trail activity, including benchmarking and robustness evaluation, will be conducted; furthermore the LRU sorting algorithm will be compared with other ordering algorithms.

Finally, we are currently working on the implementation of an intrusion detection system and a tool for traffic profiling based on the proposed monitoring architecture with the aim of demonstrating the polyhedric nature of our framework.

References

[Bace, 2000] Bace, R. G. (2000). *Intrusion Detection*. Macmillan Technical Publishing.

[Barford and Plonka, 2001] Barford, P. and Plonka, D. (2001). Characteristics of network traffic flow anomalies.

[Brownlee et al., 1997] Brownlee, N., Mills, C., and Ruth, G. (1997). Rfc 2063 traffic flow measurement: Architecture.

[Claffy et al., 1995] Claffy, K. C., Braun, H.-W., and Polyzos, G. C. (1995). A parameterizable methodology for internet traffic flow profiling. *IEEE Journal of Selected Areas in Communications*, 13(8):1481–1494.

[Han et al., 2002] Han, S.-H., Kim, M.-S., Ju, H.-T., and Hong, J. W.-K. (2002). The architecture of ng-mon: A passive network monitoring system for high-speed ip networks. *Proceedings of the 13th IFIP/IEEE International Workshop on Distributed Systems*, 2506 / 2002:16–27.

[Kitatsuji and Yamazaki, 2004] Kitatsuji, Y. and Yamazaki, K. (2004). A distributed real-time tool for ip-flow measurement. *Proceedings of the 2004 International Symposium on Applications and the Internet*.

[Mao et al., 2001] Mao, Y., Chen, K., Wang, D., and Zheng, W. (2001). Cluster-based online monitoring system of web traffic. *Proceedings of the 3rd international workshop on Web information and data management*.

[Sadasivan et al., 2005] Sadasivan, G., Brownlee, N., Claise, B., and Quittek, J. (2005). Ipfix working group internet draft, architecture model for ip flow information export.