

# Implementation of a General-Purpose Network Measurement System

Frederic Raspall  
NEC Europe Ltd.

Network Laboratories Heidelberg  
E-mail: raspall@netlab.nec.de

Andreas Kock  
T-Systems

Systems Integration, T-Systems Nova GmbH  
E-mail: a.kock@t-systems.com

## Abstract

*In this paper we report about a flexible measurement system capable of providing traffic information at different granularities and time scales, which makes it suitable to provide input to a number of management applications. We describe the different components in our prototype implementation and describe how new post-processing tools can be implemented<sup>1</sup>.*

## 1 Introduction

In order to manage the increasing complexity of today's Internet, a number of techniques are being developed to monitor, configure, enhance and troubleshoot the networks and interconnections that comprise the Net. These applications, whether automated or not, which include traffic engineering, network planning, accounting, QoS monitoring, admission control and intrusion detection, need, to some extent, the input provided by network measurements, usually at differing granularities and time scales. Traditional sources of traffic statistics are SNMP [1], flow-level data like Cisco's NetFlow [2] and packet-level data. The first one provides low-volume coarse-grained data about the volume of traffic (e.g. bytes) crossing a router's interface, while the latter gives high volume fine-grained and application-specific information. NetFlow lies in between in terms of volume and level of detail, and, if used with care, can be a valuable tool for network measurement [3].

Typically (as described in NetFlow or in the RTFM architecture [4]), measurements can be taken at the network by dedicated devices (probes) or directly at the nodes supporting metering capabilities, and exported to some other entity where the desired post-processing of measurement data can be performed and further application-specific actions can be taken. The challenges are then the ability to inspect and classify traffic at high speed, the timely export

<sup>1</sup>This work was partially funded by the EU within the EU IST INTERMON project.

of the relevant data and the storage of potentially enormous amounts of information.

The IETF IPFIX WG [5] has been established to identify the requirements [6] of these measurements and set guidelines on how those are to be taken. The focus is on the definition of an export protocol [7] that allows the reporting of flows' volume metrics together with a variable number of traffic properties in order to accommodate the exporting of aggregates of heterogeneous granularity. A well defined "standardized" exporting protocol is necessary both for the interoperability of metering devices and collecting entities and for the consistent and homogeneous interpretation, by the applications, of the measurement data.

In this paper we describe a software prototype implementation of a measurement system that consists of a metering probe and IPFIX-compliant exporter and collector. Since measurements are costly in terms of processing capabilities, storage resources and bandwidth, we target a generalized and unified measurement infrastructure capable of feeding a range of management-related applications. We identify real world scenarios and based on those present a generic way to store measurement results in a relational database, which allows post-processing applications to retrieve and share the relevant data to answer typical network questions.

Among the features of our metering probe, we find the capability of detecting the existence of flows (and therefore their measurement), according to configurable *flow definitions*. A flow definition is simply the set of features that packets must have to be considered as being part of the same flow. Thus, the meter can report, for instance, on the volume and time of per-transport protocol aggregates, of per-class-of-service aggregates, of aggregated traffic crossing an interface and ending at a specific subnet, or even of the single TCP connections within HTTP transactions. Additionally, it has the ability to allocate measurement resources (memory) to those flows utilizing the higher portion of bandwidth.

This paper is structured as follows. First we give a short overview of the IPFIX protocol, still under standardization

process. Then, we describe in section 3 the prototype implementation of our IPFIX metering toolkit. We then describe a general-purpose measurement system suitable for a variety of scenarios. We show potential scenarios where IPFIX measurements can be used and, based on them and on the information provided by IPFIX measurements, we define the structure of a database where measurements from different meters can be stored. Given that database, we describe post-processing application examples and their interaction with the database.

## 2 Overview and current status of IPFIX

The IPFIX protocol is the current effort of the IETF to define a standard way of exporting IP-based flow measurement data observed at different points at the network. The major achievements by the IPFIX WG have been so far:

- the description of a number of applications that would greatly benefit from IP-flow based measurements, and thus the need for a standard way of transferring such data to achieve a high degree of interoperability.
- the identification of the requirements that the task of transferring IP-flow measurement information has to meet, regarding, for instance, what measurement information is relevant considering today's applications needs. Some of the requirements refer unavoidably to the measurement process itself, given that this task is critical considering the current technology bounds.
- the description of a functional model identifying the elements involved in the whole process of collecting and exporting measurement data. Some of the concepts in this regard come from the previous IETF's RTFM architecture and Cisco's NetFlow de-facto standard.
- finally, the definition of a protocol for the actual process of transferring flow information to a remote location where this data can be post-processed. This protocol specification targets not only the problem of how to encode the data itself, i.e. the syntax and semantics of the different information units but also how and when this data is to be exported, considering aspects like reliability, congestion awareness and implementability.

The underlying model in IPFIX assumes that measurements are taken at the network at the so-called *observation points*, where per-IP-flow statistics can be computed by *metering processes* or meters. The *observation point* is the term used to refer to any location within a network where traffic packets can be inspected. This includes then, routers, middleboxes, traffic measurement probes attached to lines

or monitored ports or even a shared medium. Meters are the concrete entities that classify packets into flows and keep volume/time metrics corresponding to those flows.

At this point, it is important to note that IPFIX states no particular definition about what a flow is. Rather, it defines a flow as all those traffic packets traversing some point in the network with some "common" *properties*. Such properties can include values derived from the observation of packet header fields (e.g. the protocol type, the *tos* field or a portion of the destination IP address within the IP header), properties of the packet itself (e.g. its size, number of MPLS labels) or even the result derived from packet treatment (e.g. output interface in case of a router). IPFIX identifies the spectrum of properties suitable for defining flows. However, the set of packet properties that conform a concrete flow definition are open and this is what makes the protocol suitable for exporting data to be used by a range of applications. By allowing flexible flow definitions the same meter can report about traffic characteristics at very different granularities. Thus meters inspect packets, perform per-flow classification according to some "flow definitions" and update per-flow statistics at some *cache*. These per-flow properties and the corresponding volume/time values are the units of information to report on and are called *flow records*. Note that, given this flexibility, the same packet may classify as being part of more than one flow and thus cause the update of more than one *flow record* at the cache, if several flow definitions "co-exist".

Once per-flow statistics are made available at some *cache* by some metering process, they can be exported to one or more remote *collecting processes*. There the measurement information might be stored or post-processed by applications, but those issues do not fall within the scope of the protocol.

The focus of the IPFIX WG is how this transfer of information is done given a number of different constraints. If the measurements are to be taken at high-speed links, the metering process may have to inspect and classify an enormous number of packets. This may lead to either the need to keep a vast amount of per-flow data (*flow records*) requiring then big caches or to early transfer them to the collecting process, which may conflict with the requirement of the transport protocol being congestion friendly. Conversely, if a congestion-aware protocol is used, the meter may not be able to report (and then free) flow records before the cache overflows and some flows may not be measured if no memory is available at the cache. Current protocol candidates are TCP or SCTP if the transport has to be congestion-aware. While TCP is well-known, it may be too complex to implement. SCTP may be simpler, but less well-known.

Another problem is that of the heterogeneity of flow records coexisting. Flow records, in addition to vol-

ume/time values contain the properties that define the flow that these values refer to. Given the above mentioned freedom in defining flows, it is possible to have flow records corresponding to coarse-grained high-volume aggregates with a set of features like "all UDP/IPv6 traffic" or flow records corresponding to, for instance, "all TCP/IPv4 packets with destination port 8080 ending at some destination prefix 37.23.0.0/16". This new degree of freedom adds the problem of encoding records with variable-length and different number of fields and meaning. A possible solution is to use some sort of TLV encoding indicating to the collector how to parse and interpret each of the flow records within the IPFIX messages. The problem of that approach is that it is not efficient to transfer for every flow record the rule to decode it. The solution adopted by IPFIX relies on the use of templates that apply to all flow records containing the same properties (i.e. with the same flow definition). That is, before reporting on any flow record, the exporter must have reported to the collector the template needed to decode this flow record. This template indicates the fields and their lengths that further flow records are composed of. Then, flow records corresponding to this template can be transferred together indicating first the ID of the template that the collector should use to decode them. This is the way to get around the problem, but causes an additional one: that of ensuring that the templates are available at the collector, which can not be always guaranteed if an unreliable transport protocol is used.

### 3 IPFIX metering Toolkit

The core components of the measurement system that we have implemented are an IPFIX meter/exporter and an IPFIX collector. In this section we describe them and, in the following, we describe the other components that build the complete measurement system.

#### 3.1 Meter-Exporter

Our prototype of the meter-exporter consists of a Linux/PC machine with Ethernet network interfaces that behaves as a probe. The idea is to attach it to a shared medium where it can observe the traffic passing by. The software we developed runs as a process in user-space and relies on the *libpcap* sniffing library, which feeds our process with the packets (fragments) captured at one of the interfaces, which is put in promiscuous mode. As shown in figure 1, the meter/exporter has a manager which is responsible for the remote reception of *measurement tasks*. *Measurement tasks* are much like firewall rules inasmuch they specify the set of properties that packets must have in order to be considered as belonging to the same flow. So, measurement tasks are the flow definitions that the distinct

applications can request the meter to measure and report on. Every measurement task includes also the time frame when the measurement is to be performed and an indication of the application that requested it, as we shall see later on. Thus the manager holds the set of measurement tasks and activates/deactivates them when appropriate.

The *packet inspection* module performs the very first packet-processing tasks. It is invoked as a callback function from *libpcap* whenever a sniffed packet is made available. *Libpcap* provides, to the *packet inspection* module, fragments of packets together with the timestamp that the kernel puts on reception of a new packet from the network interfaces. The *packet inspection* parses the different headers within the fragment of the packet and builds a *key* which is made available to the next module.

The *filtering* module decides, for every input key, whether it is eligible for measurement, given the set of currently active measurement tasks. For each "matched" task, an integer is generated by applying a hash function to a portion of the key. This hash number, the key, the length of the packet and its arrival time (timestamp) are passed to the next module, the *flow cache*.

The *flow cache* is the set of data structures and access methods used to store and update the flow records, respectively. Every flow record contains a different number of flow properties, those according to the flow definition in the measurement task that caused the observation of the flow. In addition, each flow record contains byte and packet counters and the time when the first and last packet belonging to the flow were observed. The *flow cache* uses the hash-generated integer provided by the *filtering* module as an index to access the appropriate flow record structure. Collisions caused by the hash function are avoided by using the packet key and a list of records is kept for every entry if necessary. Once the proper flow record is found, the packet counter, byte counter and the timestamp of the last observed packet are updated. If, when a new flow is detected, the cache has no available memory to hold a new flow record, it can decide to discard an existing record, the one that experienced the least activity in the past. The algorithm we employed for that purpose is a variant of that reported in [8] for the detection of "elephant" flows.

The *exporter* is the entity that performs the transfer of measurement data to the *collector*. It accesses the cache, selects the flow records to be exported and builds IPFIX messages out of them. As mandated by the future standard, it ensures that the *template* needed to decode and interpret every flow record is exported in advance. Sets of flow records are then exported indicating first a *templateID* which is used to select, at the collector, the appropriate decoding template. Flow records are cleared, either when they do not experience any activity for a *timeout* period or if the cache runs out of memory space. The cleared records



Figure 1. Software meter layout

are first exported if they experienced activity since the last time they were exported. The transport protocol used in our prototype implementation to export IPFIX messages can be either TCP or UDP.

### 3.2 Collector

The collector is the entity that receives measurement data in IPFIX format from one or more exporters. In principle, its design has less stringent requirements than those of the meter/exporter.

Our prototype consists of a user-space process that behaves as server that can either decode IPFIX messages if they are transported on top of UDP or it can handle different TCP connections, typically one per exporter.

The collector must receive and keep a set of templates, each identified by a certain *templateID*, in order to decode all information (e.g. flow's properties) contained in the exported flow records. Since the IPFIX protocol allows for more than one exporter reporting to the same collector there is the risk that two or more exporters use the same *templateID* for different templates. That is, the uniqueness of templates/*templateID*'s that can be easily accomplished when there is only one exporter, must be enforced also when there are a number of them, each independently choosing the *templateID*'s for their templates.

Our collector waits for TCP connections and creates a

new child process whenever a meter connects to it in order to handle the reception of records and templates. This way we leave to the kernel the scheduling of the delivery of messages from the different meters and keep the *templateID* space separate, since each process uses a different memory space. This simplifies the implementation. Note that, if TCP is used instead of UDP (which is the transport typically used in previous systems like in NetFlow v.5), the transport protocol will split IPFIX messages in segments. Thus the collector cannot rely on IPFIX messages being complete nor aligned. Our collector parses as many data as available in the segment provided by TCP and buffers that which is unparseable until more data is available. Figure 2 depicts the layout of our implementation.

## 4 The complete measurement system

The meter-exporter and the collector presented in the previous section are the core components of our measurement system. In this section we complete the description of our measurement system by showing how the different applications can request the measurement of some type of traffic and how the applications can get either the results or the raw data to be post-processed.

We implemented a Java applet as a Graphical-User-Interface (GUI) so that every application can request a certain measurement by calling it. The GUI allows the user to specify the definition of the flows to be measured, together

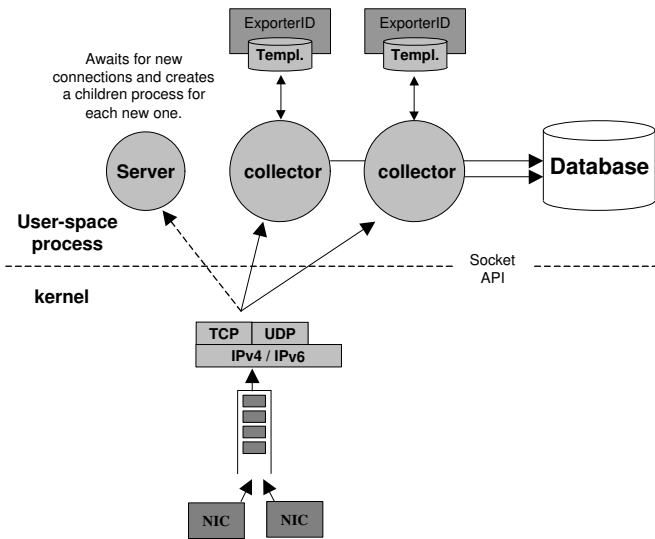


Figure 2. Software collector layout

with the restrictions for the traffic. That is, the user can filter based on some fields and consider a variable number of other fields in the definition for the detection of flows. This way, the meter will consider, for every value observed in a "detectable" field, a different flow. In addition, the GUI asks for the meter to be configured<sup>2</sup>, and the duration of the measurement. Every meter has a collector associated to it to which the results should be sent. Once all this data is configured, the GUI opens a connection to the proper meter and the manager in that meter acknowledges the successful/unsuccessful reception and acceptance of the measurement task.

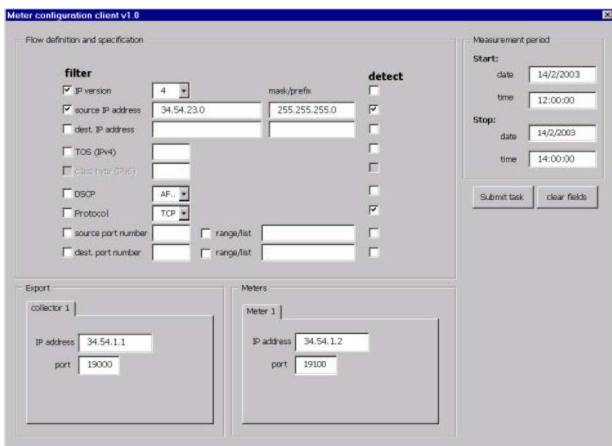


Figure 3. Graphical User Interface (GUI) for the setup of measurement tasks at meters

<sup>2</sup>selectable from a set of available meters, identified by their IP address

The last component in our system is a database(s) where the collector(s) store the measurement data, as already shown in figure 2. We chose to use an SQL database since the interaction with it is well known and easy. The different applications can easily send SQL queries and receive either the result of already processed data or the raw data to perform the post-processing themselves. The usage of a DBMS allows storing precompiled procedures in a database, which is of help in any case.

Figure 4 depicts the components of the complete measurement system and their interaction.

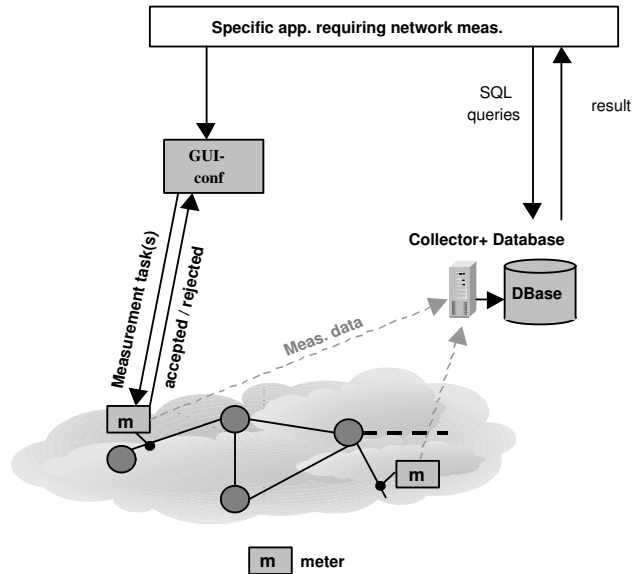


Figure 4. Complete system

A typical use of the system would involve the following cycle:

- an application requiring measurement data places a measurement task at one or more meters
- the meters perform the measurement and export the observed flows to the collectors, where they are inserted in a database
- the application can query the database if the meter acknowledged the acceptance of the request. The specific way how the database is queried, when (how often) and what data is requested depends very much on every application.

## 5 Database design

In this section we explain how to store the measurement data received at the collector in the relational database. We

explain the reasons for the chosen solution and in the following section we will explain how data can be retrieved by the applications to answer typical management questions.

In our prototype design, we took into account that:

- it should be easy for the collector to insert new data.
- it should be easy to retrieve data when a query is submitted from the applications.
- each user can place more than a measurement task at the same meter
- for each measurement task a variable (unknown) number of flows (according to a flow definition) can be observed if there are detection fields.
- each of the flows can be reported by the meter more than once, with updated volume and time values. This generates redundant data, which should be avoided: the properties of the flow, which are reported more than once.

With these considerations in mind we split the data among three tables. In the first table, called *TASKS* we store information about the measurement task. Specifically, the ID of the user that placed the measurement, the application for which it was intended (*AppID*) and the meter where the measurement took place. The *userID* and *appID* are actually derived from the IPFIX templateID field. IPFIX does not specify any mechanism to indicate the "ownership" of the measured data so what we do in our prototype is let the meter choose a templateID as a function of the *userID* and *appID*. This way the collector can invert that function and recover the *userID* and *appID*. Each measurement task is uniquely identified at the database by the *MeasID*, which is the primary key.

The second table, *RECORDS*, holds the properties fields of the flow records. Every row refers to a specific flow, which is univocally specified by the *flow record ID (FRID)* field. The table also holds the *MeasID* corresponding to the task that lead to the observation of that flow. Note that, depending on the flow definition in the measurement task, a variable number of fields are available at the *RECORDS* table for each flow.

Since the same flow may be reported several times, a third table called *VALUES* holds the actual time frame and the volume of bytes and packets observed for that flow in that interval.

As an example, consider the tables depicted in figure 8. Some network administrator (user "77") places two measurement tasks for different applications ("16" and "19") at a meter at 63.24.12.5. The first task consists of the detection of aggregated TCP/IPv4 traffic ending in the subnet 67.34.0.0/16, and aggregating it into flows depending on the "detected" source IP 24-bit prefixes. The second

task consists of the measurement of all IPv4 per-transport-protocol aggregates. That is, this task would measure the amount of per-protocol volume crossing a specific measurement point. This second task, for instance, makes the meter observe and report three flows, one for TCP, one for UDP and one for ICMP. Given the high level of aggregation of this flow definition, the meter is likely to keep the flow records at the cache since they will most likely experience activity and never be timeouted. So, most likely they will be reported more than once and this will cause the storage of a sequence of volume/time entries in the *VALUES* table. This is the case of the flow with *FRID=jkl* in figure 8.

Finally, note that having a policy on the *appID* codes, which mainly identify the type of measurement, different applications can benefit from the measurement results in the database.

## 6 Applications and use cases

In this section we describe three example contexts where our measurement system could be used to provide measurement data to management applications. It is not our intention to cover every possible of scenario. Rather, to show how applications can get the data from the database.

### 6.1 QoS scenario

Consider an ISP providing different levels (classes) of service to other customer networks. It may be required to measure then the amount of per-class-of-service volume being transmitted from a specific customer to perform, for instance, per-QoS-and-volume charging or to adapt the QoS parameters for that customer considering past usage. For example, it may be desirable to check the proportion of traffic from that customer that was marked (downgraded) to a higher drop-precedence because it exceeded the contracted rate. It may also be desirable to monitor the per-transport-protocol volume within each of the classes.

Let's consider the case that an ISP wants to see, for instance the amount of per-class aggregate that one of its customers is injecting in the network at some point *X*. It could request a meter at *X* to measure the aggregate of all traffic coming from some sourceIP prefix and to split the measurement into per-class aggregates by selecting the field *tos* as a detection field. That is, the flow definition could look, for instance, like *Flow = (IPv = 4, SourceIP = 217.13.22.x, tos = detect)*. This would lead to the reporting of several flow records from *X*, one for every *tos* value observed. Moreover, each of those flow records would be reported several times. The database could be then queried as follows<sup>3</sup>:

<sup>3</sup>we omit specific SQL queries, since there are multiple choices to get the same data

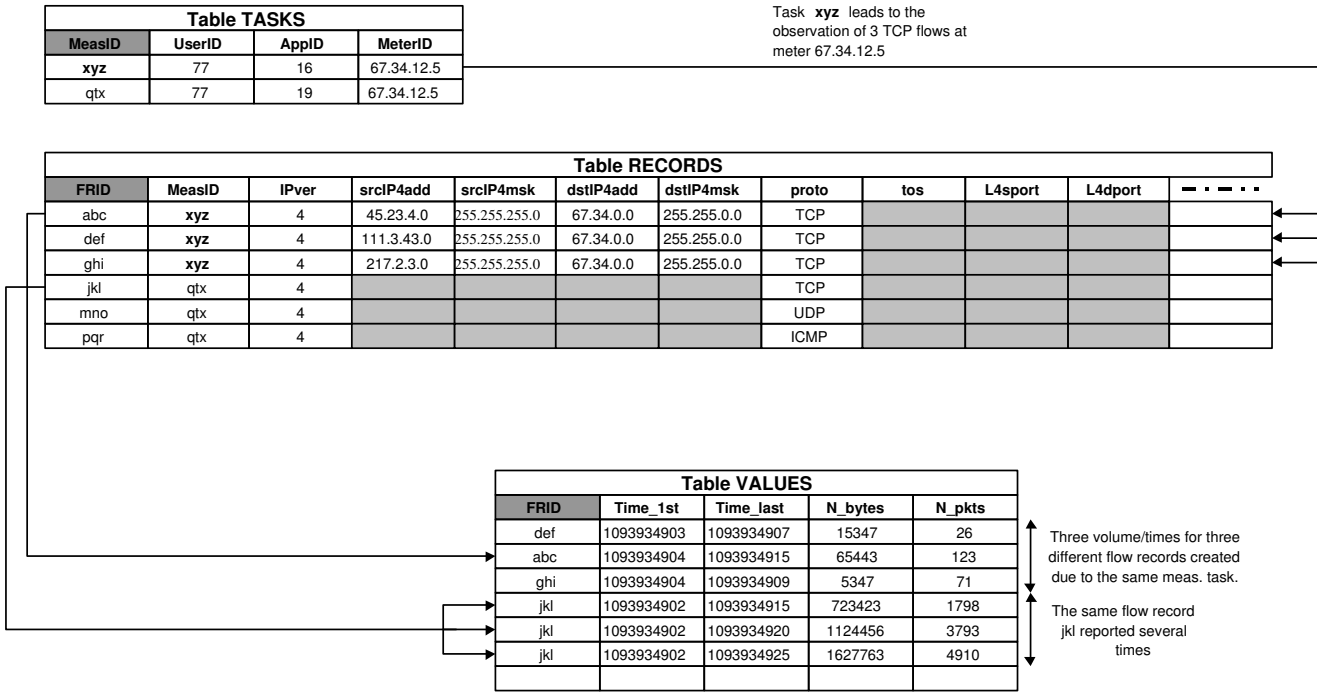


Figure 5. Tables in the relational database

from table **TASKS**

- get the MeasID corresponding to UserID=us, AppID=our\_app and MeterID=X

from table **RECORDS**

- get all flow records (FRIDs) with that MeasID

from table **VALUES**

- get the time\_1st, time\_last, N\_bytes and N\_pkts for each of the FRIDs

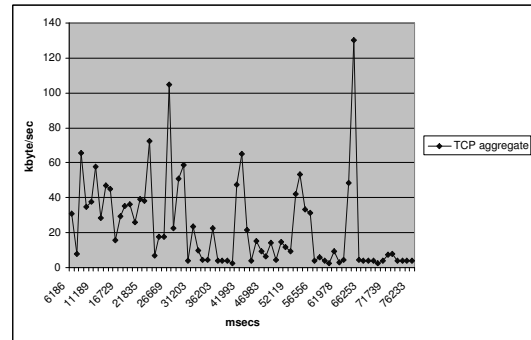


Figure 6. Instantaneous rate of a TCP aggregate measured locally at our testbed.

At the end of the query, the application would have, for each per-class-of-service aggregate observed, a sequence of time intervals  $I_i = [T_{first}^i, T_{last}^i]$  and a sequence of volume values  $V_i = (N_{bytes}^i, N_{packets}^i)$  referring to the number of bytes and packets corresponding to that aggregate in that interval. With this data, the application could derive, for instance, the "instantaneous" rate for each of the aggregates ( and compare them ), for instance by  $r_i = \frac{(N_{bytes}^i - N_{bytes}^{i-1})}{(T_{last}^i - T_{last}^{i-1})}$ . Obviously the precision of such estimate would depend on the reporting rate, which can be setup on a per-task basis. Figure 6 shows the output of a simple post-processing of a TCP aggregate that we measured at our testbed.

## 6.2 Peering control

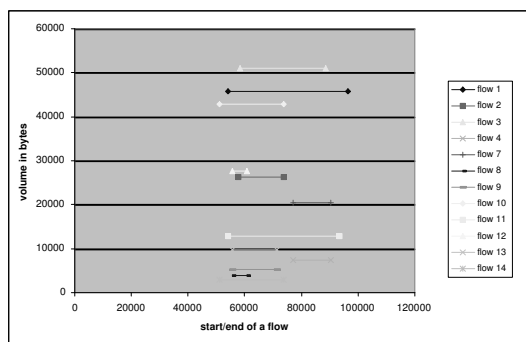
Consider two ISPs that are connected at two peering points. The ISPs have agreed, for example, that one of the peering points is to be used only for the exchange of UDP traffic (e.g. for Voice-over-IP calls), while the other is for the remaining traffic. It may be required then to find out whether this agreement is being violated or not. A possible solution would be to place a probe at the first peering point and monitor the traffic passing by, considering per-

protocol aggregates. The query of the database would be very similar to that in the previous example, and we would get volume-time metrics for each of the protocol aggregates observed in the peering point.

### 6.3 LAN-to-ISP traffic control and detection of malicious traffic

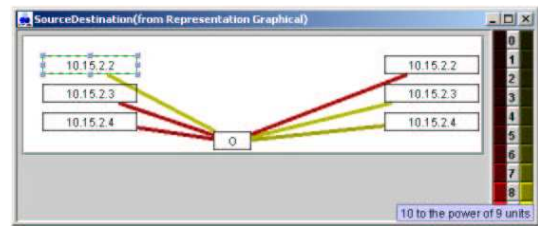
Consider a LAN connected to the Internet through an ISP. The local network has contracted a volume-based connectivity service. The administrator of the LAN may want to monitor what kind of traffic accounts for the highest amount of volume. Further, he may want to know what host is generating more traffic. A possible way to achieve this would be by means of installing a metering probe in the LAN before the NAT/Firewall.

In this context we developed a flow matrix visualization application to display what flows have been transmitted within a specific time interval and their relative volumes. This application may also be useful to detect typical DoS attacks like those based on ICMP, by measuring per-protocol aggregates and detecting an abnormal amount of traffic of this type.



**Figure 7. Different flows that compose a specific aggregate. In the X-axis the duration of the flow is shown. In the Y-axis, the total volume transmitted by the flow. We removed the description of each flow from the figures. The figure was obtained by measuring HTTP traffic.**

Another possible case is that of detecting malicious applications that scan the allowed ports in firewalls. An early detection of such hosts can be useful to add a new firewall rule to block that host. For that application, a meter behind the firewall would be required.



**Figure 8. Snapshot of the flow matrix visualization tool showing some artificially-generated flows at our testbed**

## 7 Summary

In this paper we report about a prototype general-purpose measurement system. We describe how data can be collected and exported using the currently being standardized IPFIX protocol and how a measurement system can be built on top. We describe our prototype software implementation of each of the components and how the different applications can query measurement data according to our database design. The use of a well-know database like MySQL allows to easily write post-processing applications, since many clients already exist. In fact, some of the examples shown were implemented as script files that use existing clients to access the database and that write the data into files/pipes so that the actual post-processing can be done. We are currently in the process of improving and optimizing some of the components of the system.

## References

- [1] William Stallings, *SNMP, SNMPv2, SNMPv3 and RMON 1 nad 2*, Addison-Wesley, 1999.
- [2] Cisco Systems Inc., "Netflow services and applications - White paper," [http://www.cisco.com/warp/public/cc/pd/iosw/ioft/neflct/tech/napps\\_wvp.htm](http://www.cisco.com/warp/public/cc/pd/iosw/ioft/neflct/tech/napps_wvp.htm).
- [3] Robin Sommer and Anja Feldmann, "NetFlow: Information loss or win?," *Internet Measurement Workshop*, 2002.
- [4] Realtime Traffic Flow Measurement IETF WG RTFM, " <http://www2.auckland.ac.nz/net/Internet/rtfm/>.
- [5] IP Flow Information Export IPFIX IETF WG, " <http://www.ietf.org/html.charters/ipfix-charter.html>.
- [6] B. Claise J. Quittek, T. Zseby and S. Zander, "Requirements for IP flow Information Export," *Internet draft, draft-ietf-ipfix-reqs-15.txt*.

- [7] Paul Calato B. Claise, Mark Fullmer and Reinaldo Penno, "IPFIX Protocol Specifications," *Internet draft, draft-ietf-ipfix-protocol-02.txt*.
- [8] Cristian Estan and George Varghese, "New directions in traffic measurement and accounting," *First ACM SIGCOMM Internet Measurement Workshop*, 2001.