Processing and Extending Flow-Based Network Traffic Measurements

Dissertation

zur Erlangung des Doktorgrades Dr. rer. nat.

an der Georg-August-Universität Göttingen

vorgelegt von

Sven Anderson

aus Freiburg im Breisgau

Göttingen 2009

D7
Referent: Prof. Dr. Dieter Hogrefe
Korreferent: Prof. Dr. Bernhard Neumair
Tag der mündlichen Prüfung: 20. April 2009

Contents

List of Figures

List of Tables

Chapter 1

Introduction

The Internet is constantly becoming more important for our communicative life. There is no traditional communication technology or medium that does not have its counterpart based on the Internet. Whether television, cinema, radio, telephone, letters, or print media – like books, magazines and most importantly newspapers – they all have partially or even completely been replaced by an Internet based technology for an increasing number of people. Additionally, new Internet applications that have not even been possible or affordable with the former technologies, such as instant messaging, weblogs and all the other »user generated content«, for instance, are quickly becoming an important part of our social interactions and information sources.

Hence, with every day the Internet becomes more and more important and irreplaceable. The more we depend on it, the more important its reliable and sufficient functioning becomes. Because of the decentralized organization and the »best effort« principle of the Internet, it becomes a very difficult task to achieve reliability. One of the most important instrument to make the Internet more reliable is network monitoring. It gives us an eye to see »what is going on« in the network, that is, how good the performance is, how the utilization is developing, when hardware upgrades might be necessary, where the possible weak points are, and if abusive or security relevant events are appearing, to name a few examples. Besides that, monitoring is essential for any usage based accounting. The main information source of every monitoring system are measurements, especially network traffic measurements. Therefore, a good traffic measurement infrastructure is essential for all professional network operators.

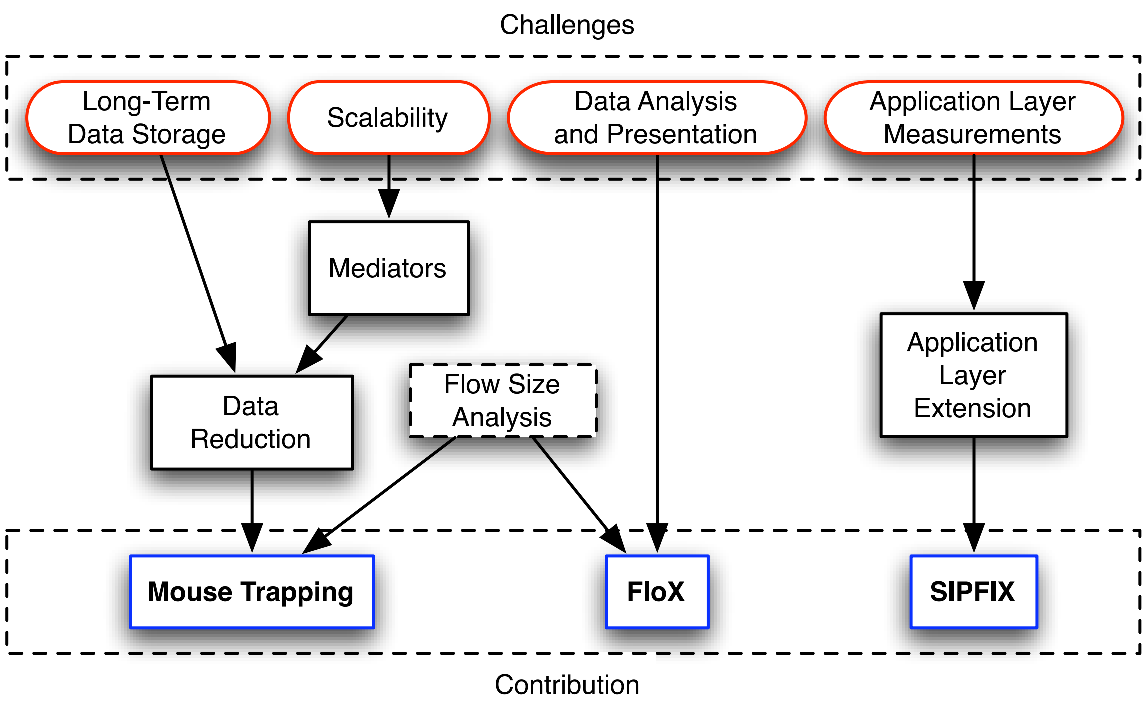


Figure 1.1: Diagram of open issues of flow-based network measurements and their relation to the solutions presented in this work.

In this dissertation, we[1] will focus on problem solutions and improvements for flow-based network traffic measurement infrastructures.

1.1 Challenges

This work mainly addresses challenges of network traffic measurements that can be categorized into two areas: the management of measurement data and the measurement of application layer information. Figure 1.1 gives an overview of the relation between open issues in current network traffic measurement technologies and the contributions presented in this dissertation.

[1]All the original work presented in this dissertation has been done by me, the author of this dissertation. However, I was always part of research groups and received helpful advices and support. Hence, the work of this dissertation has been published in papers with additional co-authors, and I will use first person plural throughout this dissertation referring to all of them.

1.1.1 Measurement Data Management

In general, there are two conflicting demands when creating network traffic measurements. On one hand, the detail of information about the observed traffic should be as high as possible. The extreme approach for achieving this would be to store the complete traffic or at least the complete headers of every packet, which is only feasible in very specific situations. On the other hand, the overhead induced by the measurement for the hardware and network links and the size of storage devices should be minimal in order to minimize performance impairments and costs. Therefore, the goal is to optimize the extraction of important information from the traffic in a concise way that is a good trade-off between detail and effort.

One approach in this direction is the very popular concept of *flow profiling*, which combines packets with the same properties into flows. This already produces a flexible compact traffic digest that can serve for many different applications. However, the rapidly increasing network speeds still lead to huge amounts of flow data, if a detail level for usual applications has to be met.

Because of these large amounts of data, it is a main challenge to build network traffic measurement infrastructures which are still feasible and scaleable. Likewise, it is important for some applications to build solutions for long-term archiving of measured data for later reference that only need reasonable amounts of data storage. A third problem comes with the vast amount and complexity of the data: a human being is not able to directly grasp it. So, in order to make use of the data, it is necessary to process, analyze and display it in a way a human observer can handle.

1.1.2 Application Layer Measurements

Another open issue is the lack of solutions for measuring packet properties that have to be extracted from the application layer. For many situations this necessary. Virtual servers, which are very common for WWW as well as e-mail, run multiple server instances on a single IP address and are only distinguished by application layer parameters. Even more importantly, various types of overlay networks are evolving in the Internet, which build their own topology on top of the transport layer, for example *Virtual-Private-Networks (VPN)*, *peer-to-peer* networks, *Voice-over-IP*, distributed content delivery networks, and so on. In order to support the monitoring of such servers and network structures, it is necessary to extend the technologies of network traffic measurements to the control plane of the according application layer protocols.

1.2 Contributions

This dissertation consists of a collection of solutions and improvements for the aforementioned challenges as shown in figure 1.1.

For the analysis and presentation of flow data the software tool *FloX* has been developed, which is designed to help an investigator interactively examine the inner structure of special events – especially high traffic peaks – with a »drill-down« approach. It enables the human observer to handle large amounts of measured data and identify the properties of the significant traffic.

The main approach to improve the scalability of distributed traffic measurement technologies is the use of *mediators*, entities that distribute the load generated by the processing of the measurement data. One of their tasks is to protect central data collectors from getting overloaded by reducing the amount of data that is being forwarded to them. Hence, mediators benefit from methods to further reduce the data by separating important from less important information. The same is true for long-term repositories of traffic data, because such reduction methods can optimize their cost-benefit ratio. *Mouse Trapping* is an attempt of such a reduction method. It exploits the fact that the large flows alone already supply sufficient information for most applications, because they represent the majority of the traffic.

It is worth noting that both FloX and Mouse Trapping are based on the analysis of flow sizes and focusing on large flows.

To include the measurement of packet properties that derive from the application layer, it is necessary to extend the concepts and technologies of the current network traffic measurements, which originally only support data extracted from the link layer up to the transport layer. Since modern flow data concepts such as in the *IPFIX* standard are extensible by design, this is possible in a consistent manner. In this dissertation, we will describe the design and definition of SIPFIX, an application layer extension based on IPFIX for one of the most important applications that builds an overlay network: Voice-over-IP networks based on the SIP protocol.

1.3 Outline

This dissertation is organized as follows: In chapter 2 we will give a condensed review of the concepts and technologies currently in use for network traffic measurements. Subjects that are important for later chapters will be emphasized. The following two chapters will deal with management of measurement data. First, chapter 3 will describe the concept of Mouse Trapping, a flow data reduction method, and evaluate its efficiency by a statistical analysis of flow sizes of real traffic data. Then, in chapter 4 the software tool FloX (Flow eXplorer) will be presented by means of design and functionality. In chapter 5 the distributed monitoring framework for VoIP networks, SIPFIX is specified in detail. After the definition of the key elements of the scheme, an extensive list of detailed showcase examples will show its functionality and flexibility. Finally in chapter 6 we will summarize the presented work and give a short outlook on possible future work.

Chapter 2

A Review of Network Traffic Measurement

In this chapter we will give a short overview of the different concepts and technologies of network measurements. First we will look into the basic concept of active and passive measurements. This will lead us to the various types of typical measurement data, especially flow profiling, which is one of the most important passive measurement concepts nowadays and the basis for the following chapters. Finally, we will look into different technologies and protocols for the transport of traffic measurement data over the network in order to allow a data collection from distributed observation points.

2.1 Measurement Methods

2.1.1 Active Measurement

The idea of *active network measurement* is to inject traffic with known characteristics into the network of interest in order to measure certain performance metrics (latency, bandwidth, jitter, ...) or structural attributes (routing tables, router links) of the observed network segment.

There are very basic active tools, which every network administrator knows from daily usage: The tool `ping` measures the *round-trip-time* to a given node and `traceroute` discovers the intermediate nodes (»hops«) and therefore the route to a given target node.

An example of a more advanced technique is the active *network tomography*, which analyzes the inner characteristics of a network by sending probe packets between outer end nodes. By examining the delay or loss rates of the probe packets it is possible to identify problems of single inner network links, to which no direct connection exists. It is also possible to analyze the topology itself and localize links and nodes which are used for many different paths and therefore have a high impact if they fail or congest. [Castro et al. 2004]

The advantage of active measurement is that it does not rely on the existing traffic. The probe packet characteristics can be freely chosen and therefore are exactly known. The measurements can be done in a systematic way in terms of time and packet properties and the results are very accurate for exactly that type of traffic that has been used as probe packets. Active measurements can also trigger active reaction of the network infrastructure, as the `ping` and `traceroute` commands show.

A drawback however is that in essence active measurements only allow conclusions about the probe traffic itself. The extent of how much this traffic represents the real payload traffic is limited and heavily depends on the specific case. For example, `ping` packets are small ICMP packets, which might get handled completely differently by the network than an RTP video stream, and the results are not necessarily transferable.

Another drawback is that active measurements always affect the observed network, since the probe traffic uses network resources itself. Consequently, what is being observed is never exactly the network as it would behave without the probe traffic.

2.1.2 Passive Measurement

Passive traffic measurement does not create any additional traffic, but observes the traffic as it passes one or more observation points. This is done either directly in the router hardware itself or with dedicated measurement hardware (»probe«), that is attached to one or more network links and receives a copy of every transmitted packet. The former solution is more cost-effective and easier to maintain, but has the drawback that the router is not only responsible for fulfilling its primary functionality, but also has to create and manage the measured data. Especially in modern high-speed networks this is often not feasible and interferes with the network stability. Therefore, in most cases a dedicated probe hardware is technically the better solution.

The obvious advantage of the passive approach is that the traffic is observed as it is and no additional interfering traffic is generated. In contrast to the active approach using artificial traffic, this is the only method to analyze for what and how the network is actually being used. But this is also the drawback at the same time, since the approach cannot gather information about situations as long as they do not appear. If a certain question is of interest, the only solution is to wait until an event happens that allows the answering of that question.

Because of the fundamental differences of passive and active measurements, in general the active approach is mainly used to answer questions like »how does the network look like and in which status is it currently«, while the passive approach is mainly used to answer questions like »what happens in the network and for what is it used«. Of course, there is an overlap of these two areas and there is no clear distinction, since they heavily depend on each other. This is why most network monitoring solutions make use of both active and passive measurements to create an overview of the network by combining the results of each.

This dissertation focuses mainly on the processing and management of data gathered by passive measurements, although the VoIP monitoring framework presented in chapter 5 also includes the report of performance metrics which are possibly gathered by active measurement methods.

2.2 Measurement Data Formats

Usually traffic measurement solutions have to serve not only a single but a number of several purposes, such as

- giving an overview of the current network performance and status [Manajan et al. 2001],
- feeding Intrusion Detection Systems,
- archiving usage statistics for billing [Duffield et al. 2001],
- giving long term trends for planning of network upgrades [Feldmann et al. 2001] or efficient routing [Shaikh et al. 1999], and
- providing data for investigation of network problems.

To keep traffic measurements versatile, it is desirable to keep as much information about the observed traffic as possible. The naïve and extreme approach to keep a copy of all packet headers or even the whole transfered data – what would be without doubts the most versatile solution – is in most cases not an option for obvious reasons.

It is therefore inevitable to extract information from the raw traffic data, keeping the amount of data relatively small while the information is still usable in a flexible way. A trade-off has to be found between the conciseness and the flexibility of the extracted information. And it depends very much on the application and the specific needs where this trade-off lies.

In the following sections we will describe two important basic concepts of collecting and storing measurement data, *time series* and *flows*, which represent such trade-offs with different emphases.

2.2.1 Traffic Traces

In some cases it is necessary to take the extreme approach and store the complete headers of every single packet or even the complete traffic, which is called *traffic traces*. This of course generates an extreme amount of raw data and therefore is not a method used on a regular basis in network monitoring solutions. It is mainly used for a limited period of time when a special event or problem has to be examined in-depth. Therefore it is mostly common in the academic field or for special forensic examinations. Since simply all data is stored, traffic traces provide the maximum flexibility.

Traffic traces are typically created with standard PC hardware and software like tcpdump [tcpdump] or with specialized hardware-accelerated network measurement cards, such as those by the DAG project [DAG]. The raw data is usually stored and distributed in so-called *pcap-files*.

2.2.2 Time Series

If one-dimensional metrics have to be observed over time, it is useful to record time series of the relevant values. Therefore the values are continuously recorded for a series of time intervals. These values can either be directly observed by sampling or averaging a *gauge*, or they derive from increasing *counters*. In the latter case the discrete derivative with respect to time of the counter is used (counter steps per time interval) by either taking the difference of the current and the previous counter values or by resetting the counter to zero after each measurement. For network monitoring, typical examples of such values are:

- bandwith usage (counter)
- delays and round-trip-times (gauge)
- system load of routers/hosts (gauge)
- events per time (dropped packets, special packets, ...) (counter)
- number of parallel clients/nodes (WLAN, mobile networks, ...) (gauge)
- number of parallel flows/connections (gauge)

An important advantage of time series is that they can be easily displayed as time graphs, which show the development of the value during a certain time interval. This is an efficient way for human observers to catch a lot a of information in a short moment and to get an overview of the situation.

Even several time series can be displayed in a single graph without necessarily overloading the observer. An example is given in figure 2.1. It shows the network traffic bandwidth of a number of selected port numbers over time, that is, of all packets which have the given port number either as source or destination port. The time series are stacked upon each other, which gives the observer an impression of

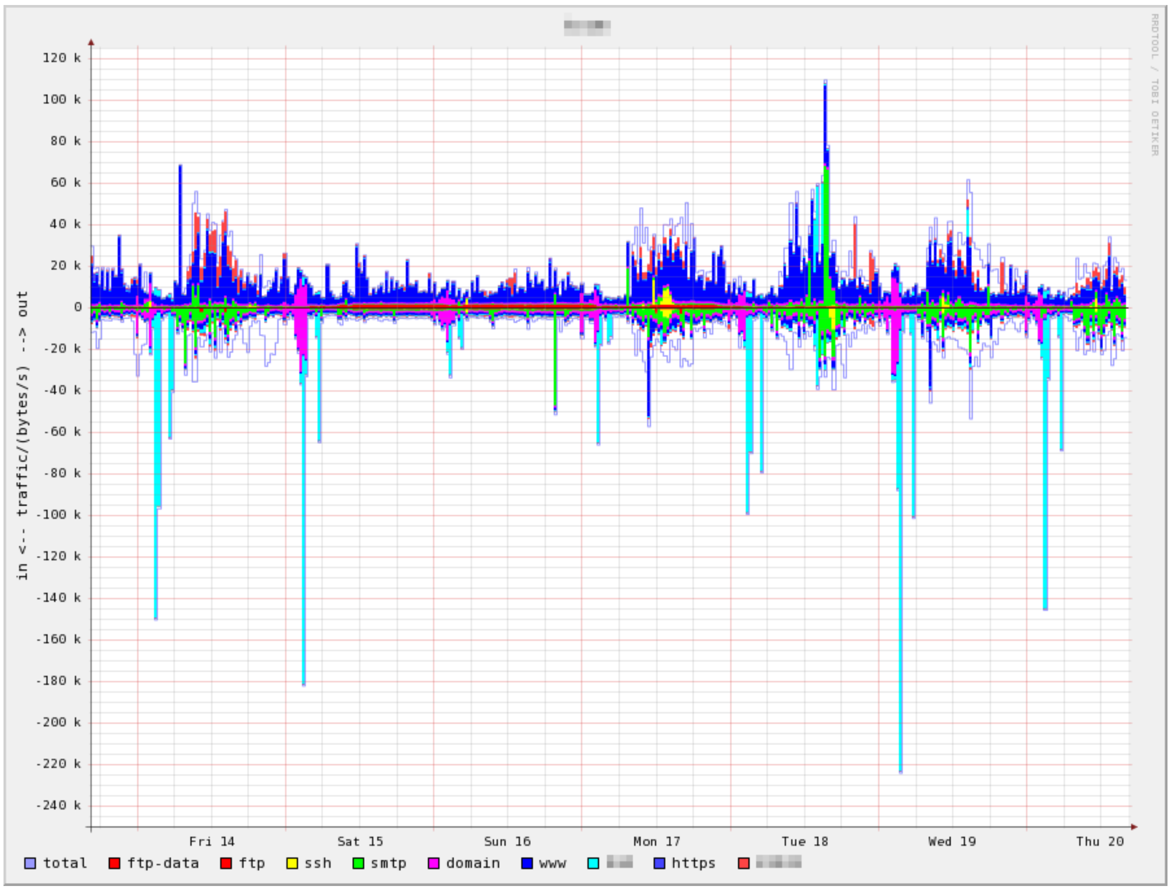


Figure 2.1: Example of time series measurements recorded and displayed with rrdtool: the transmitted bytes per second for certain ports are shown in a stacked plot.

the overall traffic and also what ports (and therefore which applications) this traffic consists of at any given time.

In order to obtain both a fine time resolution and a long time span of recorded data, it is a common approach to keep several time series in parallel with different time resolutions. The most recent time period, such as the last minutes or hours, are available in the finest resolution, while other records with less time granularity can span over very long time windows like months or even years.

Time series are a very efficient way to store traffic data in many cases, even in more complex cases like the example above, where the observed traffic is separated in a number of smaller traffic subgroups. However, the criterion by which the data is separated for an independent processing has to be known beforehand. The time series themselves do not contain any information about the packets, so a separation afterwards or on-demand is not possible anymore.

On the other hand, every additional separation criterion increases the number of necessary time series by an factor of the number of possible criterion values. Therefore, the number of time series increase exponentially with the number of separation criteria, which is easily becoming inefficient. In typical cases two time series for incoming and outgoing traffic are created for the traffic of each host and/or each port. But for more complex and flexible traffic data storage, time series are usually not appropriate and a more flexible data format is necessary.

2.2.3 Flows

If a more flexible data format than simple time series is needed, the most common approach to extract and store data about network traffic is the concept of *flows*. For flow based measurements all packets are categorized according to a set of certain packet properties. Which properties belong to such a set can be freely defined according to the needs. A single flow according to such a definition comprises all the packets that share the same values for all the defined properties. For each observed flow, that is, a flow of which packets have been observed, an entry in a flow table is made. For each flow different measures can be taken separately, such as the number of bytes, number of packets, number of TCP connections and time-stamps of the first and last observed packets, for instance.

An architecture based on the concept of traffic flow measurement has been defined in [RFC 2722] by the Real-Time Flow Measurement (RTFM) Working Group of the IETF.

A common example for a flow definition is given by the 7-tuple of the following property fields:

- `Source IP address`
- `Destination IP address`

- `Source port` (for UDP or TCP)
- `Destination port` (for UDP or TCP)
- `IP protocol`
- `Ingress interface`
- `IP Type of Service`

It is the standard flow definition initially used by *NetFlow* [NetFlow], a very successful protocol by Cisco based on the flow concept, which exports measured flow data from routers to central collectors for storage and analysis. Because of the broad deployment of Cisco hardware in network infrastructures, NetFlow had a strong influence on the development of flow based network technologies.

But of course a flow definition can also look different, for example a simple pair

- `Source IP address`
- `Destination IP address`

or an even less detailed pair

- `Source ASN`
- `Destination ASN`

which defines very coarse flows including all packets coming from and going to the same *Autonomous Systems*, represented by the Autonomous System Number (ASN), and might be an appropriate definition for backbone operators that are not interested in single end-node resolution.

Modern flow concepts, like *IPFIX*, which will be described in more detail in section 2.3.3, are designed to allow a flexible definition of flows. The packet properties used for the definitions are usually values extracted from packet headers, but are not necessarily limited to these. They can also derive from the packet handling, like the value `ingress interface` in the example above shows, or from any other characteristic of the packets.

The observed flows are stored in data tables, together with the measured counters (usually number of bytes and packets) and time-stamps, as *flow records*. This is done in regular intervals, depending on the desired time resolution. Again a decision has to be made between conciseness (longer time intervals) and the detail of information. The more often flow records are stored and the shorter the time intervals are, the more detailed is the information about the dynamics of the traffic. A single flow record, with only one counter value for its whole time interval, cannot tell the difference between a bursty and a continuous bandwidth usage, for instance. However, by using time-stamps for the first and last observed packet of a flow record, the represented time window can be minimized, which increases the time accuracy. For the same reasons it is not possible to decide in general if two identical consecutive flow records represent two »transport sessions« or a single persisting one.

ip_src	**ip_dst**	**port_src**	**port_dst**	**proto**	**tos**	**pkts**	**bytes**	**time**
192.168.171.139	192.168.27.69	1409	80	6	0	23	1623	2006-06-28 14:45:00
192.168.27.69	192.168.171.139	80	1409	6	0	33	41272	2006-06-28 14:45:00
192.168.163.254	192.168.216.39	234	33491	6	0	1	40	2006-06-28 14:45:00
192.168.202.138	192.168.242.5	80	42811	6	0	8	1692	2006-06-28 14:45:00
192.168.8.100	192.168.27.2	59855	53	17	0	1	59	2006-06-28 14:45:00
192.168.27.2	192.168.8.100	53	59855	17	0	1	158	2006-06-28 14:45:00
192.168.132.155	192.168.27.123	57764	80	6	0	6	620	2006-06-28 14:45:00
192.168.27.123	192.168.132.155	80	57764	6	0	5	1775	2006-06-28 14:45:00
192.168.132.155	192.168.27.123	57765	80	6	0	38	2393	2006-06-28 14:45:00
192.168.27.123	192.168.132.155	80	57765	6	0	66	90691	2006-06-28 14:45:00
192.168.171.139	192.168.27.69	1410	80	6	0	9	950	2006-06-28 14:45:00
192.168.27.69	192.168.171.139	80	1410	6	0	8	6998	2006-06-28 14:45:00
192.168.150.16	192.168.80.73	57021	25	6	0	50	57065	2006-06-28 14:45:00
192.168.80.73	192.168.150.16	25	57021	6	0	44	2825	2006-06-28 14:45:00
192.168.217.76	192.168.110.98	25	42883	6	0	2	88	2006-06-28 14:45:00
192.168.197.95	192.168.80.73	1028	25	6	32	4	260	2006-06-28 14:45:00
192.168.110.98	192.168.217.76	42883	25	6	0	1	77	2006-06-28 14:45:00
192.168.217.2	192.168.33.130	1053	53	17	0	1	69	2006-06-28 14:45:00
192.168.80.12	192.168.110.98	59128	25	6	0	5	300	2006-06-28 14:45:00
192.168.33.130	192.168.217.2	53	1053	17	0	1	120	2006-06-28 14:45:00
192.168.80.73	192.168.166.198	59129	25	6	0	9	471	2006-06-28 14:45:00
192.168.44.95	192.168.80.31	4787	25	6	0	3	128	2006-06-28 14:45:00
192.168.80.31	192.168.44.95	25	4787	6	0	1	48	2006-06-28 14:45:00
192.168.162.206	192.168.216.36	0	0	1	0	6	168	2006-06-28 14:45:00
192.168.216.36	192.168.162.206	0	0	1	0	6	168	2006-06-28 14:45:00

Table 2.1: Examples of flow records with a 6-tuple of property fields, packet and byte counters and a time-stamp of storage time.

An example of such a set of flow records is given in table 2.1. In this case the flows are defined by the 6-tuple of the fields

- `Source IP address (ip_src)`
- `Destination IP address (ip_dst)`
- `Source port (port_src)`
- `Destination port (port_dst)`
- `IP protocol (proto)`
- `IP Type of Service (tos)`

and include the counters

- `number of packets (pkts)`
- `number of bytes (bytes)`

as well as a time-stamp for the storage time marking the end of a time slot represented by each flow record.

A common misunderstanding is that flows are usually *not* bidirectional, that is they contain only the packets passing the observation point in *one* direction. The packets traveling in the opposite direction have different source and destination values and therefore belong to a different flow. Concepts exist, though, to combine the correlated flows into so-called *Biflows* [RFC 5103].

Also, a flow is generally not synchronized with transport layer sessions. Consecutive TCP connections with the same flow-relevant property values belong to the same flow, and can usually not be distinguished. Of course it is possible to include a property that represents a kind of a »session identifier«, which would separate each session into single flows, but this is not common.

2.2.4 Packet Sampling

In the past link speeds has been increasing by about 50% each year [Roberts 2000], while the speed of cheap memory (DRAM) only increased by about 10% per year [Patterson and Hennessy 1998]. As a result the gap between link speeds and memory increased as well. DRAM is too slow to keep up with the counters of flows in modern network infrastructures. At the same time the number of parallel flows are increasing with the faster link speeds, and it is impossible to keep all of them in expensive fast memory (SRAM).

This is the reason why packet sampling has been introduced as a network measurement technique. Instead of capturing and analyzing every single packet on the line, only some packets are selected. From this sub-population of selected, »sampled« packets estimates are made about the total population of packets.

There are different approaches for basic sampling methods:

every Nth: This is the simplest approach and is how sampling with Cisco NetFlow works. It strictly captures every Nth packet in a regular manner. Although this is easy to implement and has a low performance profile, it can lead to aliasing effects in case of regular patterns in the observed traffic.

1 out of N: This approach randomly selects one packet out of N packets. It introduces randomness and thereby reduces the aliasing effects, while it still maintains a constant sampling rate of one per N packets.

sampling probability $p = 1/N$: Every packet is sampled with a probability of $p = 1/N$. This approach has the least aliasing effects. On average one packet out of N is sampled. However, the sampling rate is not constant but variable, and there is a relatively high probability that bursts of consequently sampled packets occur. The capturing hardware and software and its buffers must be able to handle most of these bursts.

The information of the sampled packets can either be stored as special forms of flow records, which represent only a single packet, or they can be used to estimate the counters of the flows to which the sampled packets belong to [Duffield et al. 2003].

Besides these basic methods there are more sophisticated and specialized methods being developed. For example, [Estan and Varghese 2003] propose a *Sample and Hold* algorithm that samples packets with a probability p, but then captures all subsequent packets that belong to the same flow as the first sampled packet. This is possible because the per flow counters are held in a relatively small amount of fast SRAM. This method identifies the largest flows and has a higher accuracy in respect to normal sampling.

Another variant of sampling is *deterministic sampling*. In this case the selection function is a function of a packet property. A common example is to apply a hash function on the header or the payload of the packet and to select packets with certain hash values. This way a random-like packet selection can be achieved, while the selection is still fully deterministic. That is, if different measurement points use the same selection function, they will all sample exactly the same packets, which can be important for comparison reasons, like for measuring one-way-delays from one measurement point to another.

An IETF Working Group has been build for standardizing the methods and the data handling of packet sampling in network measurements, called PSAMP [PSAMP]. The documents produced by this Working Group define a standard set of sampling selection operations and how the information about the sampled packets and the sampling process itself can be stored and exchanged in an interoperable way. This Working Group is working closely together with the IPFIX Working Group, which developed a flow information exchange standard that will be described in section 2.3.3.

2.3 Distributed Data Collection

Usually, the traffic observation has not only to be done at a single observation point, but at a whole set of distributed observation points. On the other hand, it is necessary or at least helpful to have all the data at a single central point in order to process, analyze, display and archive it in a structured and unified way, giving the network operators the possibility to get an overview of the network as a whole.

Therefore, it is necessary to transfer the measured traffic data to central data storages in a standardized and interoperable way. There are a couple of different protocols in use to transfer or access network traffic data, mainly depending on the type of traffic data. In the following sections we will introduce the most important and common ones.

2.3.1 SNMP

The Simple Network Management Protocol (SNMP) is a standard protocol defined by the IETF [RFC 3411]. It is designed to make information available about the configuration and status of network-attached devices and systems. It is a *polling* protocol, which means that a managing application accesses the device and asks for the information it is interested in whenever it needs it.

Routers by most manufacturers have internal byte and packet counters, which can be accessed via SNMP along with other network related information. These counters are usually polled by central monitoring applications. SNMP values are the usual input for all time series based measurements as described in section 2.2.2, since they offer a simple interface to the counters or gauges which can be polled in a regular interval to build up the time series databases.

Although very useful and common for a general overview and simple statistics, the use of SNMP values is very limited. While SNMP is designed to transfer single values on demand, it is not able to transfer more complex multidimensional measurements like flow tables and the like.

2.3.2 NetFlow

In order to transfer measured traffic flow data from different measurement points to a central collection point, Cisco developed *NetFlow* [NetFlow]. Besides being a feature name for Cisco products that support flow measurements, the term »NetFlow« also refers to the protocol that is being used to transfer the data the routers and probes collected in their flow tables. In contrast to SNMP, NetFlow is not a polling but a *pushing* protocol. That is, the data measured and collected over time is actively sent to central servers, called *Collectors*.

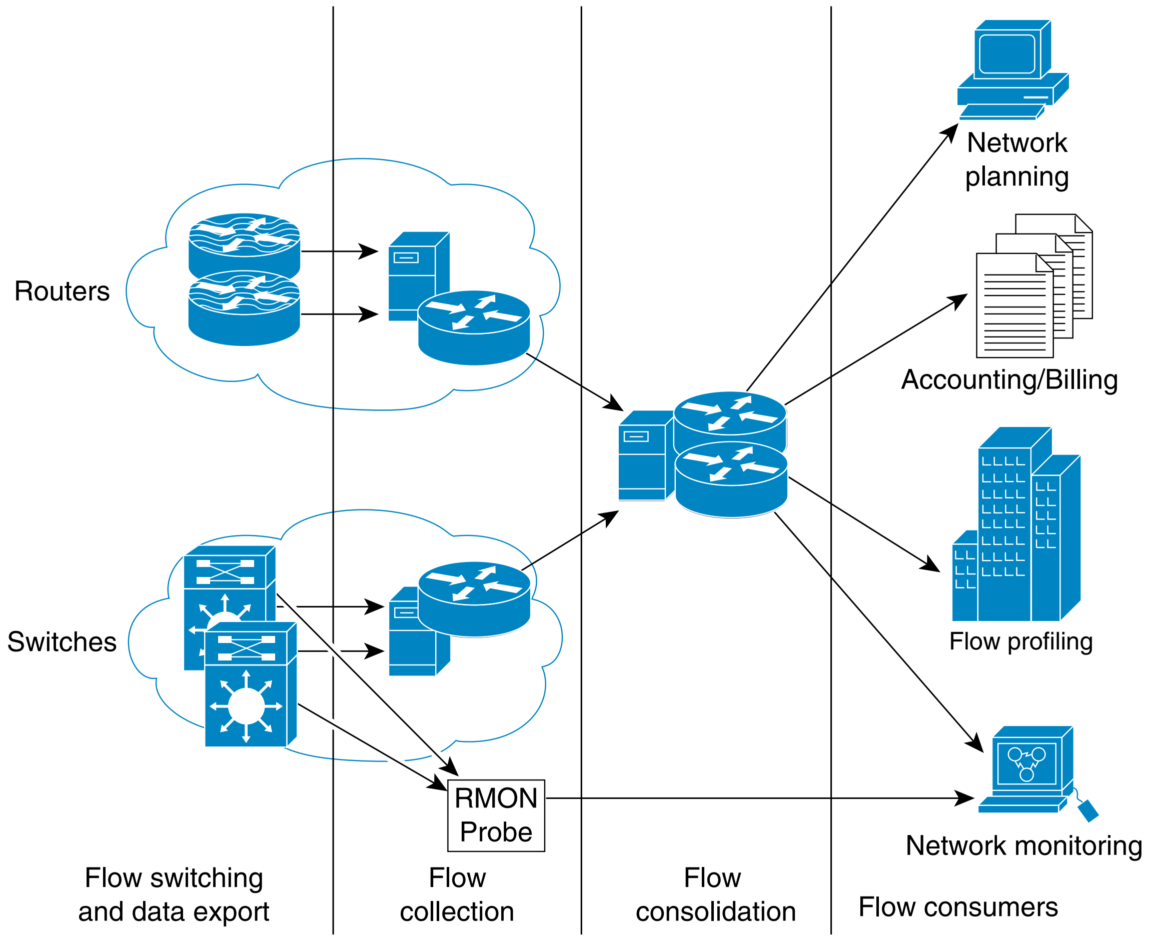


Figure 2.2: Architecture of distributed network measurements based on NetFlow. (Diagram by Cisco)

In figure 2.2, the basic architecture of a NetFlow-based network measurement setup is shown. Several NetFlow devices export flow records via the NetFlow protocol to the Collectors. These NetFlow devices are either routers that directly observe the traffic they forward, or probes that are connected to mirroring ports of central switches. The Collectors are responsible for storing the data they receive and for making the data available for further processing, analyzing and displaying by flow applications.

Except in most recent versions [Flexible NetFlow], the flow definition of NetFlow is fixed and consists of the 7-tuple given in the example on page 13.

Because of the dominance of Cisco products in network infrastructures, NetFlow has become the de-facto standard for the transfer of flow-based network traffic measurements and also has been adopted by other manufacturers. It is still the most common technology in today's distributed measurement setups.

2.3.3 IPFIX

Although NetFlow is a widely adopted technology, it still remains a proprietary standard. There was the need for an open internet standard for exporting flow data of observed IP traffic that guarantees interoperability between exporting and collecting devices of different manufacturers. Therefore, an IETF Working Group for IP Flow Information eXport (IPFIX) has been established in order to create such a standard.

After evaluation of different technologies, NetFlow version 9 has been chosen as the basis for the IPFIX standard. Because of the wide deployment of NetFlow solutions, it can be expected that the IPFIX standard will soon be widely accepted, since the similarity to NetFlow makes the migration fast and cost-effective.

2.3.3.1 Architecture

In figure 2.3, the reference model of the IPFIX architecture is shown. Like with NetFlow, there are two functional units: *Exporters* of Flow Records, which are called *IPFIX Devices*, and *Collectors*, which receive the Flow Records. The flow data consuming applications are either directly incorporated into the Collectors, or are separate entities which communicate with the Collectors. The communication between applications and Collectors is not part of the IPFIX standard.

IPFIX devices consist of two functional blocks, the *Metering Processes* and the *Exporting Processes*. Metering Processes are responsible for the following tasks:

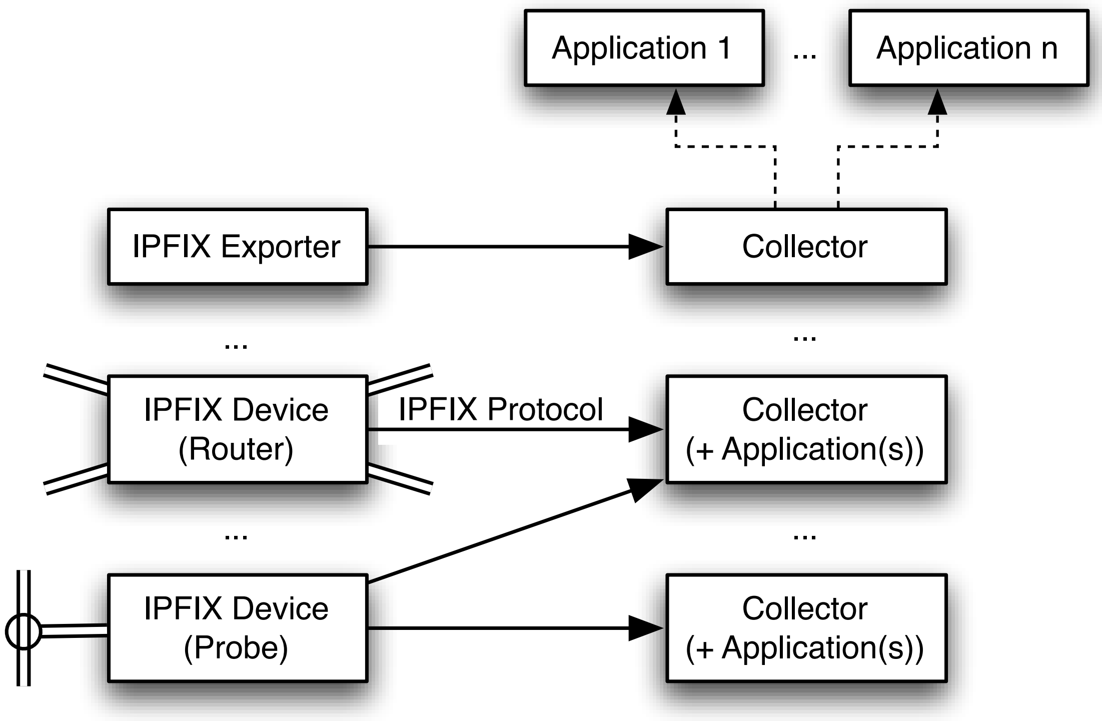


Figure 2.3: Reference model of the IPFIX architecture.

- capturing packet headers from one or several *Observation Points*, which together build an *Observation Domain*,
- time-stamping,
- sampling (optional),
- filtering (optional),
- creating and managing a flow table according to given flow definitions,
- timing out of inactive flows,
- handling of resource overloads.

The Exporting Processes are responsible for:

- creating Flow Records from the flows that are listed in the flow table,
- deciding when to send out Flow Records and related messages,
- creating IPFIX messages in compliance with the IPFIX protocol,
- sending out IPFIX messages to one or several Collectors.

Collectors host one or more *Collecting Processes*, which receive the flow data from one or several IPFIX Devices and have to take care of the following:

- receiving and decoding of the Flow Records and other IPFIX messages,
- storing and managing of meta data of the IPFIX protocol (*Control Information*),
- storing of the Flow Records,
- making available the Flow Records to flow consuming applications.

2.3.3.2 Protocol

The IPFIX protocol has been defined in [RFC 5101]. The main improvement in respect to NetFlow is the use of *templates*. The use of templates has two important advantages. First, they allow a flexible definition of which fields are part of a Flow Record. This avoids Flow Records including any unnecessary fields, and their structure can be adapted to exactly that of the desired flow definition. Second, the templates allow Flow Records, which can be large amounts of data, to be transmitted without any structural overhead in a condensed binary format.

For each type of exported Flow Record a template has to be sent to the Collector in advance. The template defines the structure of the Flow Records with an ordered list of *Field Specifiers*. A Field Specifier specifies what type of data the field contains and its length in bytes. The description of the data type is done with *Information Elements (IE)*, which are defined in the IPFIX information model [RFC 5102]. Information Elements give the Collector information about the data encoding and the semantic meaning of fields. Some common examples for Information Elements and their data types are:

- `sourceIPv4Address (ipv4Address)`
- `destinationIPv4Address (ipv4Address)`
- `protocolIdentifier (unsigned8)`
- `sourceTransportPort (unsigned16)`
- `sourceMacAddress (macAddress)`
- `vlanId (unsigned16)`

- `flowStartMilliseconds (dateTimeMilliseconds)`
- `octetTotalCount (unsigned64)`
- `packetTotalCount (unsigned64)`

An important feature of the Information Element concept is its extensibility. The standard IEs can be extended by including them in the IANA IE registry. Besides that, it is possible for enterprises to register an enterprise identifier which allows them to define their own enterprise-specific Information Elements.

2.3.3.3 Flow Keys

Some fields in a Flow Record build the flow definition, that is, these fields represent the packet properties which have the same values for all packets in the same flow. These fields are called *Flow Keys*. Besides Flow Keys, a Flow Record includes other fields, most importantly measured properties of the flow as a whole, like counter fields (like `octetTotalCount` or `packetTotalCount`) or time-stamps (like `flowStartMilliseconds`).

2.3.3.4 Bidirectional Flows

RFC 5103 [RFC 5103] defines a method to export associated bidirectional Flows (*Biflows*) in a single Flow Record. Two Flows combine to a Biflow if all non-directional fields directly match, and all source-related fields match the corresponding destination-related field of the other Flow. The Flows are merged by adding special IEs for counter fields of the »reverse« direction from the destination to the source.

This has several advantages: In most cases it is more efficient to assemble Biflows at the measuring device than in a Collector. Also, Biflows share much information, so exporting them as individual Flows creates a large amount of redundant data. Furthermore, it is possible to give the two directions an additional meaning by a *Direction Assignment*. In the case of TCP connections for example the normal counters could refer to the packets sent by the node that initiated the connection, and the reverse counters to the packets received by it.

2.3.3.5 Flow Transformation

Because of the flexibility in the definition of flows, it is also possible to process IPFIX Flow Records and transform them into other flows. It is, for example, a common approach to reduce the amount of Flows Records by *flow aggregation*. This means that groups of flows are merged into a single flow by adding up their counter values. This can be done in two ways:

Aggregation by time Flow Records of the same flow but of different subsequent time periods are added up and replaced by a single Flow Record that spans over a longer time period. This is equivalent of reducing the time resolution of the Flow Records. For example, if five-minute Flow Records are aggregated to one-hour Flow Records, up to twelve Flow Records are replaced by one Flow Record.

Aggregation by Flow Keys In this case, a new flow definition is made by removing one or more Flow Keys from the definition of the existing Flow Records. All flows that only differ by the values of the removed Flow Keys are aggregated. Besides the number of flow-records, this also reduces the number of »dimensions« of the flows, that is, the number of Flow Keys. Therefore, it saves memory space in two ways. For example Flow Records with the Flow Keys `<sourceIPv4Address, destinationIPv4Address, sourceTransportPort, destinationTransportPort>` could be aggregated to Flow Records with the Flow Keys `<sourceIPv4Address, destinationIPv4Address>`. This would aggregate all the Flow Records of parallel connections between two hosts into one Flow Record.

Another important method to reduce the size of the Flow Records, but in a lossless way, is to reduce the redundancy. It is the normal case that many Flow Records are very similar and differ only in a few Flow Keys. This redundant data, the *Common Properties*, is normally included in every single Flow Record. In order to save this memory, [RFC 5473] describes how to export the Common Properties only once in a special Flow Record, called *Common Properties Data Record*, defining a `commonPropertiesID`. All the subsequent Flow Records which match these Common Properties are then replaced by Flow Records – the *Specific Properties Data Records* – that only include a single Flow Key with the corresponding `commonProp-ertiesID` instead of the original shared Flow Key fields.

2.3.3.6 Mediators

A general problem with flow-based distributed network traffic measurements is the lack of scalability. With increasing size of the observed network, the number of observation points increases as well, and with increasing link speeds, the number of parallel observed flows increases. As a result the data that has to be processed by a single central collector will easily reach existing limits of the processing or network resources of the collector. To prevent this, the IPFIX Devices would have to export less and less data to the collectors as a compensation, rendering the the whole measurement infrastructure less effective. This renders the original IPFIX architecture of IPFIX devices directly exporting to Collectors unfeasible for large and fast networks.

In order to cope with this problem, the concept of a *Mediator* is currently being developed by the IPFIX Working Group in several document drafts [Kobayashi et al. 2008a, b]. In figure 2.4 an exemplary scenario of an IPFIX infrastructure with Mediators is shown. A Mediator is basically a device which incorporates both a Collecting Process and an Exporting Process as well as with optional Intermediate Processes. It receives Flow Records from IPFIX devices or other Mediators and can process the data in a number of different ways, such as

- data reduction by aggregation or filtering,
- data correlation and combination from different devices,
- data modification (anonymization for instance), and
- data storage in distributed – and therefore scalable – repositories.

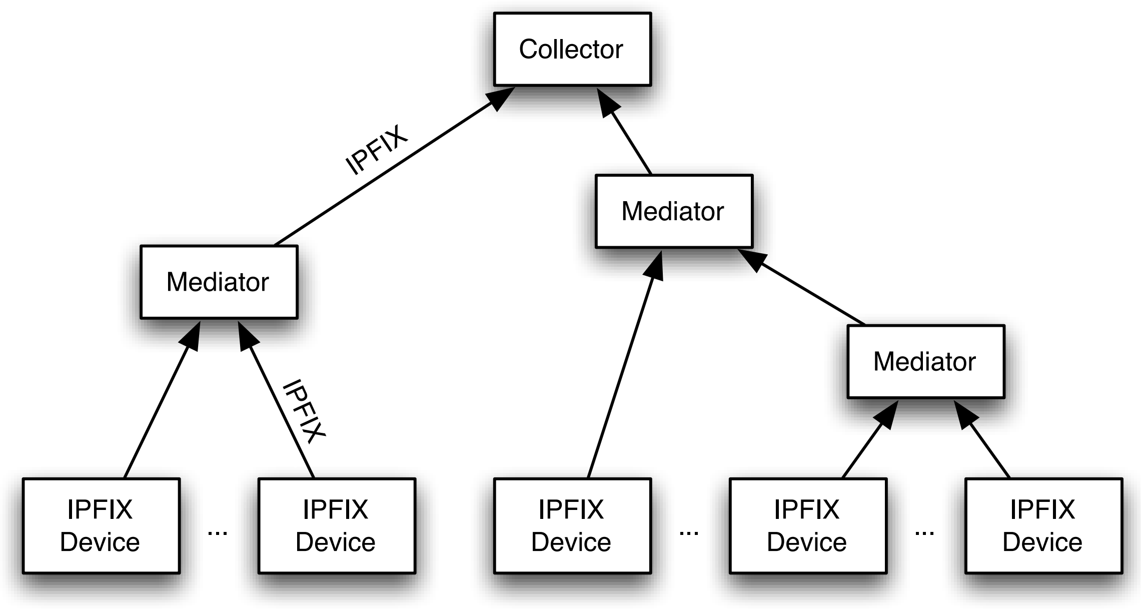


Figure 2.4: Scenario example for the use of Mediators in an IPFIX measurement infrastructure.

The processed data is exported to a Collector or another Mediator.

So, Mediators implement a tradeoff, distributing the processing and storage needs of fine grained data, while the Collector still receives the amount of data it can handle.

Chapter 3

Flow Data Reduction: Mouse Trapping

3.1 Introduction

The concept of network flows is today the most common way to accomplish passive traffic measurements for general purposes and to further process and store that data. Such flow measurements are used for numerous applications like traffic engineering, performance monitoring, detection of different types of anomalies and accounting. For some of these applications it is necessary to observe the traffic at fine granularities on a short-term basis, because anomalies may have very subtle signatures and a real-time detection and response may be required. On the other hand, for other applications the data has to be stored for a long time, to be able to observe long-term trends, or for forensic research after an incident, for instance. As a result, network operators have to deal with huge amounts of data, which implicate several problems in the data management. In most cases the size of the data repositories will not increase faster than the amount of collected traffic data, so every data repository will eventually overflow, if old data is not dealt with. Another problem which already exists today, is central data collectors cannot handle all the information from the measuring points in large networks. Therefore, to keep the whole system scaleable, so-called mediators have to be used, which, among other things, have to filter and aggregate the data, forward only the necessary data to the collectors, and act as decentralized data repositories. In order to cope with these problems, it is very helpful to have methods that are able to condense flow data, that is, to extract the more important and remove the less important data. In this chapter, based on the work in [Anderson and Hogrefe 2008], a flow data reduction method – *Mouse Trapping* – is introduced and evaluated.

Of course it depends on the application, what is considered important and what is not. But the lossy nature of such a method is not necessarily in contradiction with all the applications outlined before if it is applied at a certain time or a certain location. For instance, fine grained data, which is only kept for an optimal intrusion detection, does not have to be kept longer than the typical reaction time of the detection system.

As it has been reported by other studies like [Fang and Peterson 1999] and [Feldmann et al. 2001], for many applications it is sufficient to only look at the large flows. This is due to the fact that the flow sizes of internet traffic are highly non-uniformly distributed: many flows are very small, and very few flows are large. These applications include network engineering [Feldmann et al. 2001], detection of denial-of-service attacks and even billing [Duffield et al. 2001]. Following this, [Estan and Varghese 2003] proposed to »focus on the elephants, ignore the mice« and developed algorithms to achieve this directly in high-speed routers. But because this is quite a complex method, it is doubtful, if it will be broadly adopted.

Mouse Trapping uses the same approach, but instead of filtering the data already in the monitoring device, it applies the concept *after* the measurement in order to reduce and condense the flow data, and therefore is supposed to be used in Mediators and Collectors. Accordingly it reduces the number of flow records by filtering (»trapping«) the records of small flows (»mice«) for further aggregation or removal, while most of the traffic is still represented by the large flow records that remain. In a theoretical simulation we show, that in the range of realistic parameters this method is very effective. However, it is assumed that the non-uniform flow size distribution of internet traffic follows an underlying power-law, comparable to many other power-laws that have been identified in the field of computer networks such as: Ethernet traffic [Leland et al. 1995], WWW traffic [Crovella and Bestavros 1997], internet topology [Faloutsos et al. 1999], visits of websites [Adamic and Huberman 1999], to name a few. The evaluation of *Mouse Trapping* with real internet traffic verifies this assumption and shows that – for the simple case that all small flow records are just discarded – a reduction of 1:10 can be achieved, while information about only 5% of the traffic is lost.

3.1.1 Related work

In [Aiello et al. 2005] a different lossy compression method for traffic data is presented, which is based on signal processing techniques, which makes it difficult to see what data is actually lost. Flow size distributions have also been analyzed in [Liu and Huebner 2002] for different application protocols, but since no evaluation in respect to power-laws has been done, it cannot help to estimate the efficiency of *Mouse Trapping*. Also [Kumar et al. 2005] analyzed flow size distributions for different protocols to estimate the distributions from sampled traffic data.

3.1.2 Outline

This chapter is organized in the following way: In section 3.2 we briefly describe some terminology and statistic basics which the method is based on. Section 3.3 examines the relation between the flow sizes and the amount of traffic for simulated power-law distributed flows and introduces the *Mouse Trapping* method. In section 3.4 the method is evaluated with real traffic data for the total traffic and some application based subsets by first examining the power-law nature of their flow size distributions and finally determining the reduction efficiency.

3.2 Basic principles

3.2.1 Flows

Flows are the most common way to store general information captured by passive traffic measurements today. As the term *flow* is used in many different ways by the internet community, a short description of the flows used in this work will be given.

This is done according to the terminology and flow definition of the *IPFIX Protocol* specification [RFC 5101].

Flows are defined by a number of *flow keys*, which represent certain properties of the packets. All packets sharing the same flow key values belong to the same flow. Additionally, certain counter keys are kept per flow, for example the number of packets, the number of bytes, or the number of sessions. These flows are stored in *flow records* including time stamps of events like the first and last observed packet. This is done periodically to obtain the flow development over time.

3.2.2 Power-laws

Many objects and events in nature, technology and society follow a *power-law distribution*

$$y = Cx^{-a}, \tag{3.1}$$

where y is the quantity and x the size of an observation. There are many popular power-law distributions: The *Pareto distribution*

$$Pr\left[X \geq x\right] = \left(\frac{m}{x}\right)^{k},$$

describes the probability, that the size X of an observation is larger than x, and the *Zipf distribution*

$$X(r) \sim r^{-b},$$

describes the relation between the size X and the size-rank r of an observation. All these distribution's exponents are related by

$$a = 1 + k = 1 + b^{-1},$$

and hence they basically all describe the same type of distribution [Adamic 2000]. These distributions have the interesting property that small elements are very common and large elements are very rare. Which is why they are also known as *heavy-tail distributions*.

3.3 Mouse Trapping

In the field of computer networks and the internet many power-laws have been identified as well [Leland et al. 1995; Crovella and Bestavros 1997; Faloutsos et al. 1999; Adamic and Huberman 1999], and it is a widely accepted fact that the sizes of flows are usually highly non-uniformly distributed, that is, most flows are small, and very few flows are large [Fang and Peterson 1999; Feldmann et al. 2001].

3.3.1 Flow size distributions

If the flow sizes of usual traffic are in fact power-law distributed, the vast majority of the flows are very small flows, whereas large flows are very rare. A rising question is then, what the relation is between the amount of traffic represented by the large flows and by the small flows, respectively. This leads to one to the following assumption: The few large flows together represent the main part of the overall traffic and the many small flows represent only a small fraction of the traffic. We will look into this with more detail now.

For a numeric estimation we assume the flow size distribution

$$f(x) = Cx^{-a}$$

perfectly follows a power-law (equation 3.1), where x is the flow size, and $f(x)$ is the number of flows of size x, then the *flow size-traffic distribution*, that is the amount of traffic represented by the flows of a given size x, follows as

$$t(x) = Cx^{-a} \cdot x = Cx^{1-a}.$$

So in a double-logarithmic plot, the slope of the amount of traffic is more shallow by one than that of the number of flows, that is, the traffic is shifted to larger flow sizes in respect to the number of flows. The effect of this becomes more clear, if we look at the relation of the *cumulative distribution functions* (CDF)

$$F(x) = C_\mathrm{F} \sum_{i=x_\mathrm{min}}^{x} i^{-a} \quad \text{and} \quad T(x) = C_\mathrm{T} \sum_{i=x_\mathrm{min}}^{x} i^{1-a},$$

which are the sums of the number of flows and the amount of traffic, respectively, for all flows with a flow size $\leq x$. C_F and C_T are chosen so that the CDFs are normalized, that is, $T(x_\mathrm{max}) = F(x_\mathrm{max}) = 1$. In other words: the relation between $F(x)$ and $T(x)$ tells you what fraction of the smallest flows represent what fraction of the total amount of traffic. This relation highly depends on the exponent a and the bandwidth of flow sizes, defined by the minimum and maximum flow sizes x_min and x_max, between which the flow sizes are distributed.

3.3.2 Flow size filter

Figure 3.1 shows a normalized plot of $F(x)$ against $T(x)$ for simulated distributions for different exponents a. In this case a flow size bandwidth of 2^{26} (almost 8 decades) was used, which corresponds to the observed flows from section 3.4. It is striking that if you look at the slope for $a = 2.0$, for example, 85% of the flows represent only 10% of the overall amount of traffic. For lower a's this relation drifts even further apart. For $a = 1.6$, 95% of the flows represent only 0.6% of the traffic.

So far, no assumptions on the flows' definitions were made, that is, which specific flow keys they are composed of. We just assumed, that the flow sizes follow a power-law, which has been reported to be true for most flow data. Consequently it is very likely that for any given flow data repository it is more or less true, that most of the traffic is represented by a few large flows.

This leads to the following concept: it must be possible to reduce the number of flows by a large amount, and thus the size of the flow data in general, by filtering or aggregating (»trapping«) all small flows (»mice«) up to a certain flow size threshold X, while at the same time keeping the full information about the large flows, representing the main part of the overall traffic. For example, all the small flows

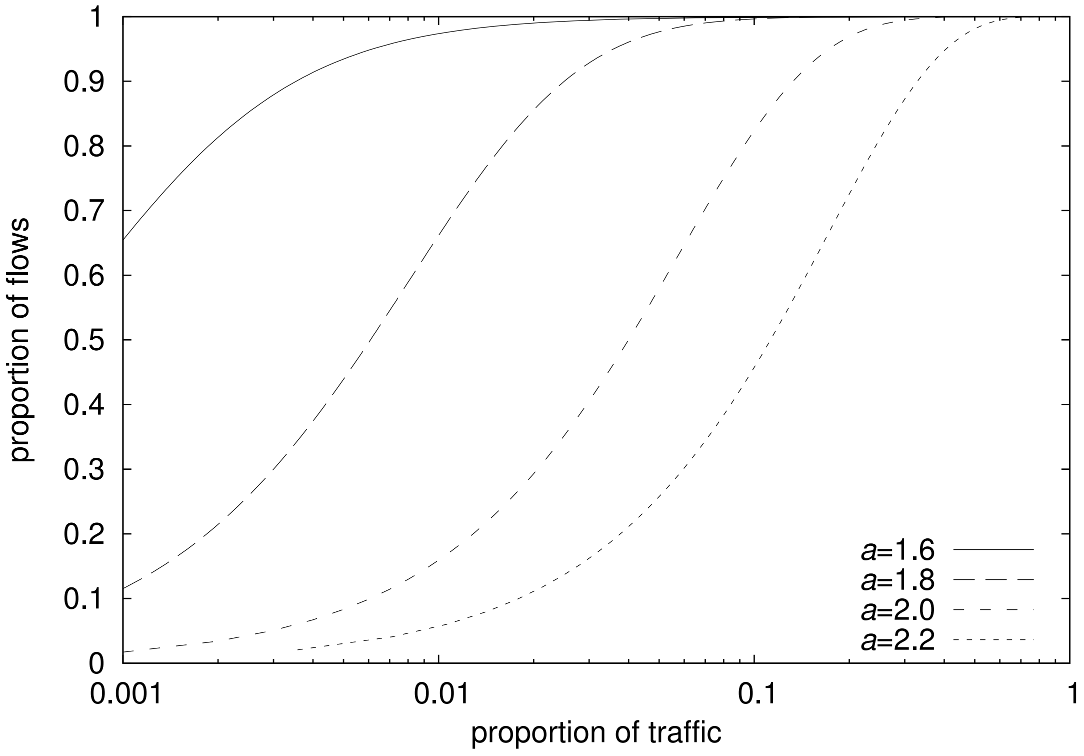


Figure 3.1: Theoretical relation between cumulated flows and cumulated traffic for different exponents a.

could be aggregated into one »noise« flow, containing no information but the total amount of the small flows. This approach – »focus on the elephants, ignore the mice« – is already quite popular in similar areas [Estan and Varghese 2003], since for many applications the knowledge of the large flows is sufficient.

The threshold X can be chosen depending on the needs of the application. If a guaranteed fixed *reduction rate* r in respect to the number of flows is needed, X can be calculated by maximizing x for which $F(x) \leq r$. (For example $r = 0.9$ to reduce to 10% of the original flow repository size.) If it is important to limit the loss of information about the represented traffic, X can be determined by a fixed *loss rate* l. In this case X is calculated by maximizing x for which $T(x) \leq l$. (For example $l = 0.1$ to lose only the flows of 10% of the traffic.) Of course X can also be estimated as a dynamically changing value, if the method is applied in a system, where an early decision has to be made, if a flow belongs to the elephants or the mice.

If this method is applied to the overall traffic with one static threshold X, it should work as expected. But there will be certain subsets of flows (like certain application protocols), which tend to have smaller packets and smaller flows and therefore will be over-proportionally affected by the loss. To avoid this, for each subset of interest a separate X_n can be calculated and applied. This makes sure that the constraints defined by r or l are true for each subset, and not only for the overall traffic.

It is up to the implementation, how the trapped small flows will be further processed. Besides the extreme solution mentioned before, in which all the »mice« are just deleted or aggregated into a single »noise« flow, many other approaches are imaginable. For example, small flows could be aggregated to bigger compound flows with less flow keys or less time resolution. In the end, even several levels of different thresholds can be combined, to keep less information the smaller the flows are. The possibilities are manifold.

3.4 Evaluation

In this section we evaluate the proposed method with real traffic data. Therefore we first analyze the data, to verify that the flow distributions in fact meet the assumption of following a power-law distribution. Then we calculate the reduction rates for different given loss rates for the simple case that all the small flows are aggregated to one single »noise« flow. To give an example how to apply the method on different subsets and to learn more about the characteristics of different applications, we will calculate this individually for certain application protocols as well.

3.4.1 Traffic data description

The measured flows in this case are classical flows, which consist of the following five flow keys:

- IP source and destination address
- source and destination port (if present)
- IP protocol ID

For each flow, the *bytes* and *packets* are counted and stored once per minute together with a time stamp in a flow record. In this case, no session information like TCP sessions or UDP packet timeouts are used to trigger flow record creation. For later data analysis, all the flow records during one hour for each flow have been aggregated into one flow record. That is, for each flow represented by a flow record the duration is limited to one hour and therefore the size is limited by $bandwidth * 3600$s. Flows of longer duration are split up into several flow records.

The traffic data has been collected from three network up-links of a medium-sized IT service and consulting company [SerNet]:

- 100 Mbit/s Ethernet
- 2 Mbit/s leased line
- 6 Mbit/s ADSL link

The data was collected for the period of 66 days from June 27th, 2006 until September 1st, 2006 and represents an overall traffic of 2.2 TiB.

For the measurement probe, a normal PC hardware was used, running OpenBSD and connected to a Gbit monitoring port of a switching hub that connects to all of the three up-links. The monitoring software in use is *pmacct*[pmacct], which in this case writes the observed flows into a PostgreSQL database.

The mixture of traffic derives from broadly diversified applications, as these links are used both to access the in-house server-farm from outside as well as by the employees for the everyday service and management work.

A detailed overview of the proportions of the flow numbers and data amount in relation to the overall traffic is given in Table 3.1 for a selection of application protocols. HTTP here refers to HTTP and HTTPS (ports 80 and 443), and FTP refers to ports 20 and 21, so FTP data connections on unprivileged ports are not

Protocol	Byte %	Flow %
HTTP	44.6	34.6
FTP	0.5	0.1
SMTP	6.5	16.7
DNS	1.0	35.2
SSH	29.3	5.9
OpenVPN	1.6	0.0
other	16.4	7.4

Table 3.1: Protocol distribution of bytes and flows

included, as these are only detectable by application layer analysis. It becomes evident that most of the flows are produced by the protocols HTTP, DNS and SMTP.

3.4.2 Distribution analysis

Extensive analysis, if the flow size distribution follows a power-law, and if so, with which exponent and error, is hard to find, especially for fine-grained flows like the ones used in this work. It is also an open question to what extent such analysis is transferable to other kinds of traffic, as *the* characteristic traffic probably does not exist.

Since the proposed method to reduce the flow data is based on the assumption that the flow sizes are distributed at least close to a power-law distribution, it was necessary first to examine and verify this for the actual data. To get a deeper insight, make the results better transferable to other types of traffic and to show how this method can be applied separately to different subsets, this is done for each application protocol from Table 3.1 independently as well as for the overall traffic.

3.4.2.1 Logarithmic binning

Figure 3.2 shows the raw flow size distribution of the overall data in a double-logarithmic plot. Although one can clearly see a linear relation between the flow

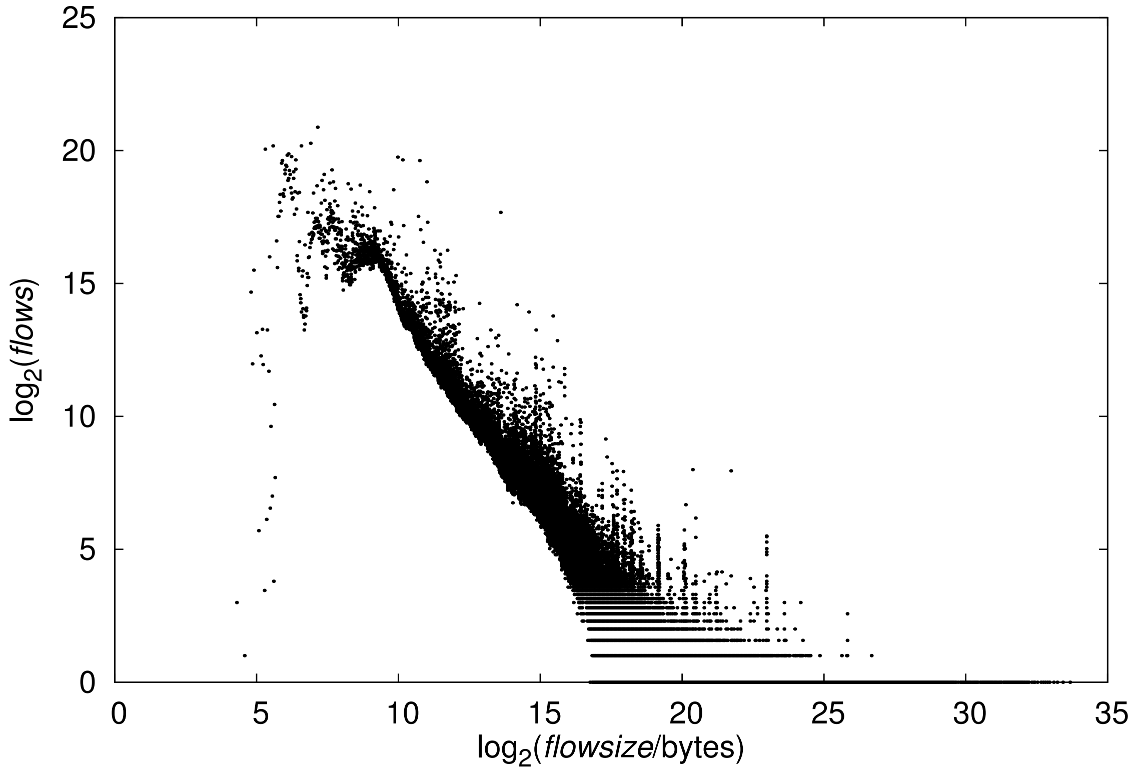
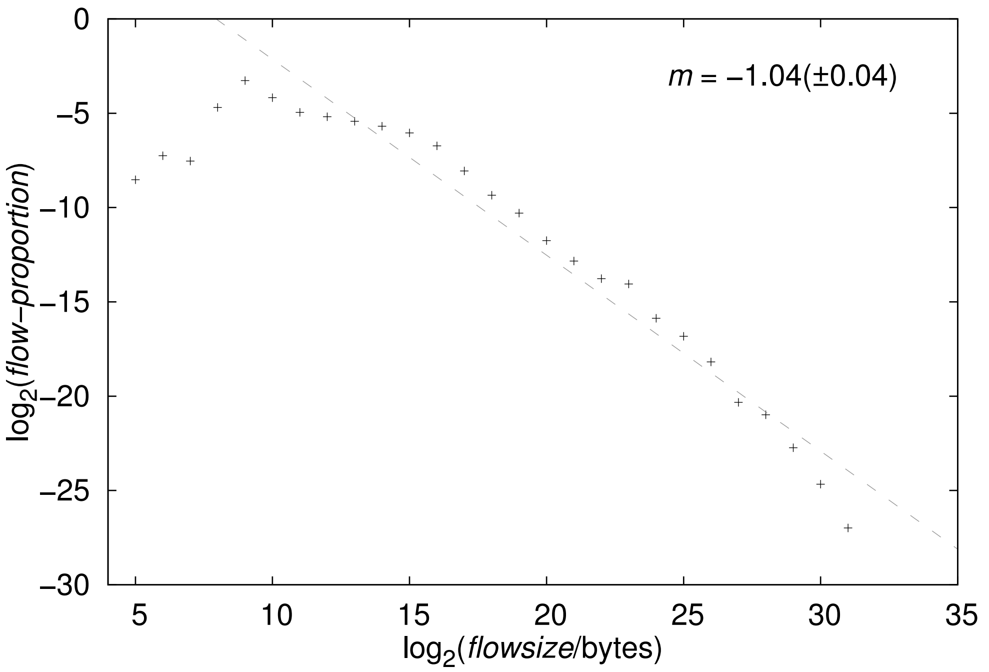


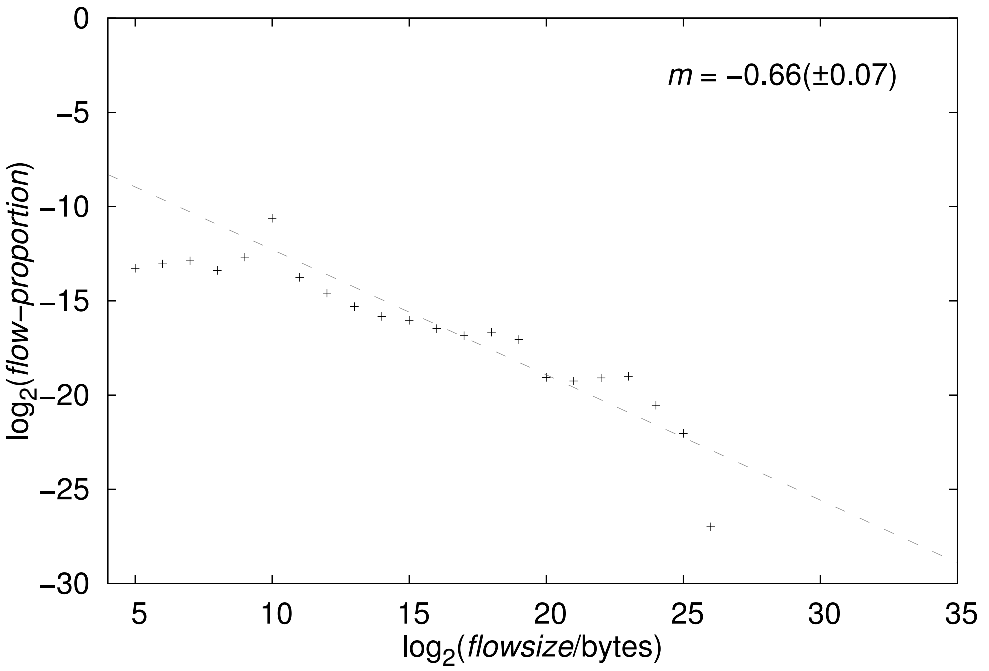
Figure 3.2: Raw data of a flow size distribution

sizes and the corresponding number of flows, the plot is very noisy, and not suitable for a linear regression analysis. This is especially true for the flattened-out tail, which biases the slope of the fitted line.

To gain a proper fit, the number of flows has to be summed up into bins of exponentially increasing flow size width, which is called *logarithmic binning*. This significantly reduces the noise and extends the region of linear relation. But because of the increasing width, the bins of higher flow sizes are weighted stronger. Applied on a power-law distribution this results into an exponent increased by one. Hence the expected slope of the fitted line is $1 - a$, which equals the exponent $-k$ of the Pareto distribution.

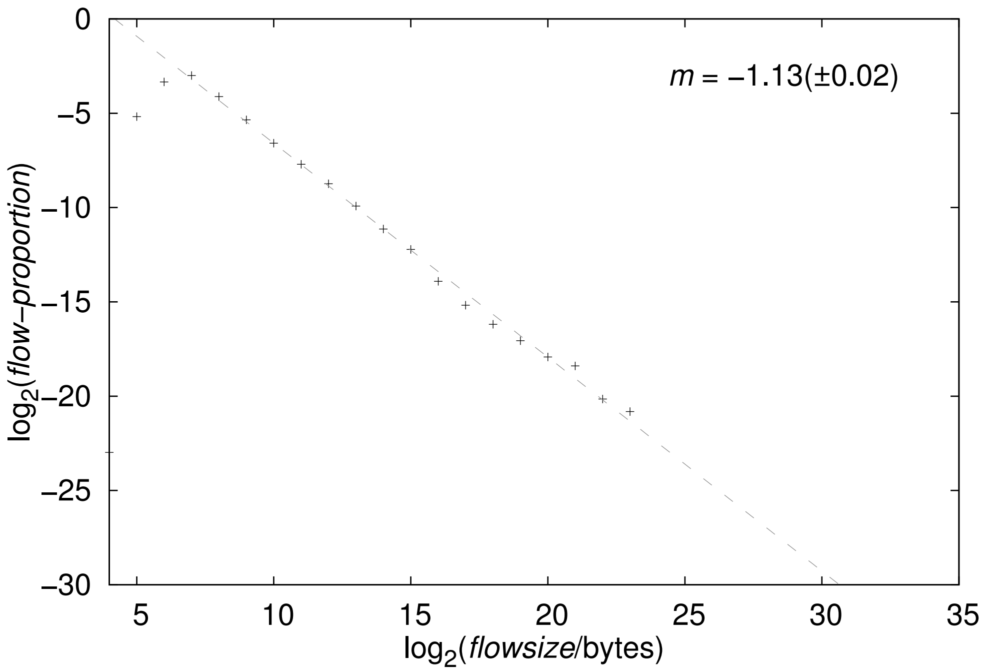


(a) HTTP

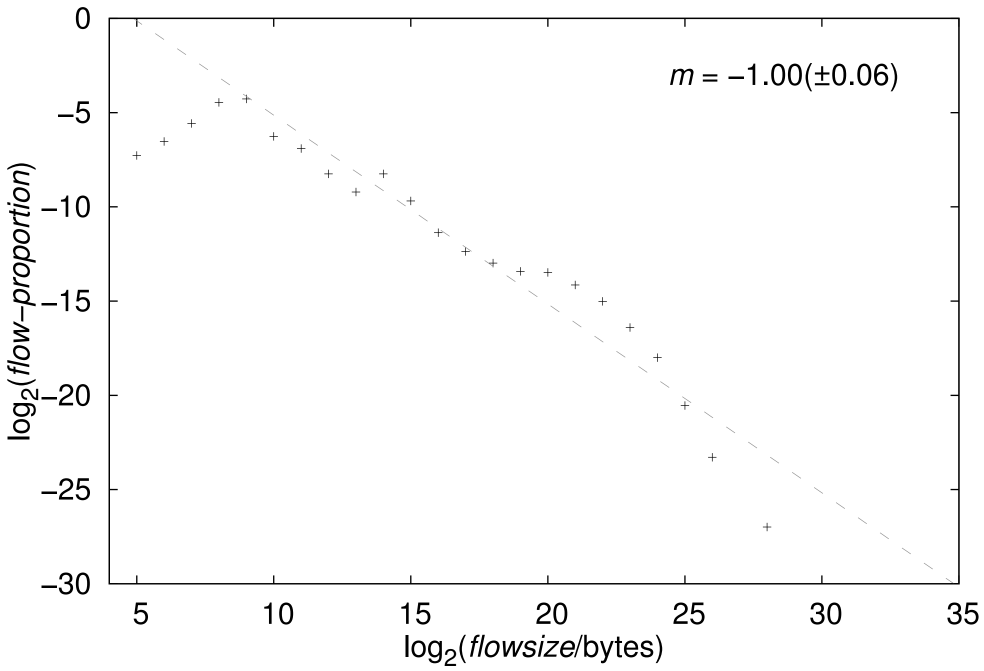


(b) FTP

Figure 3.3: Logarithmic binned flow size distributions for selected application protocols.

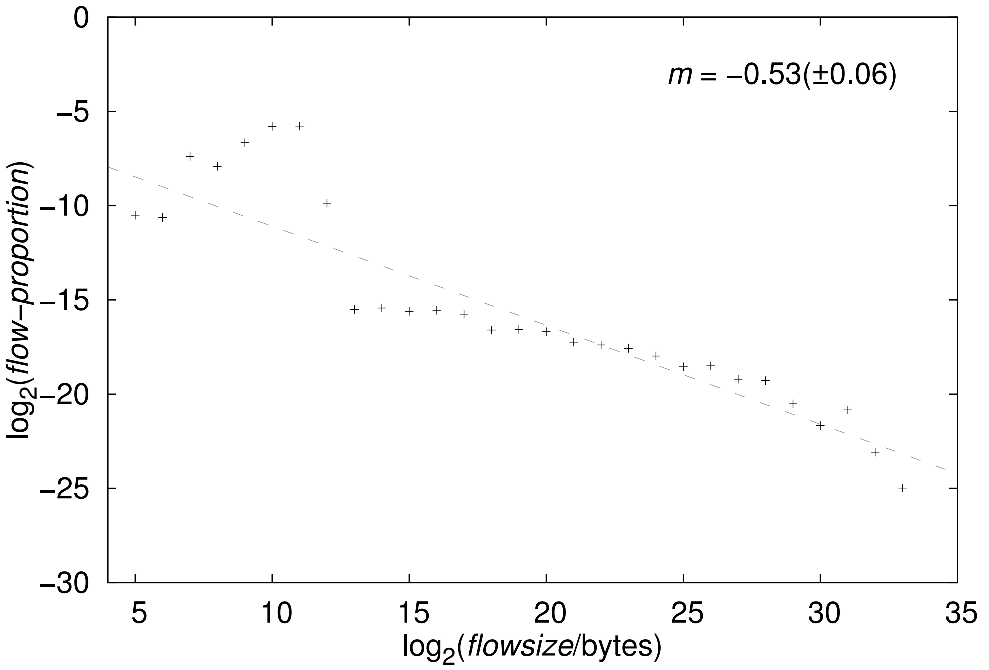


(c) DNS

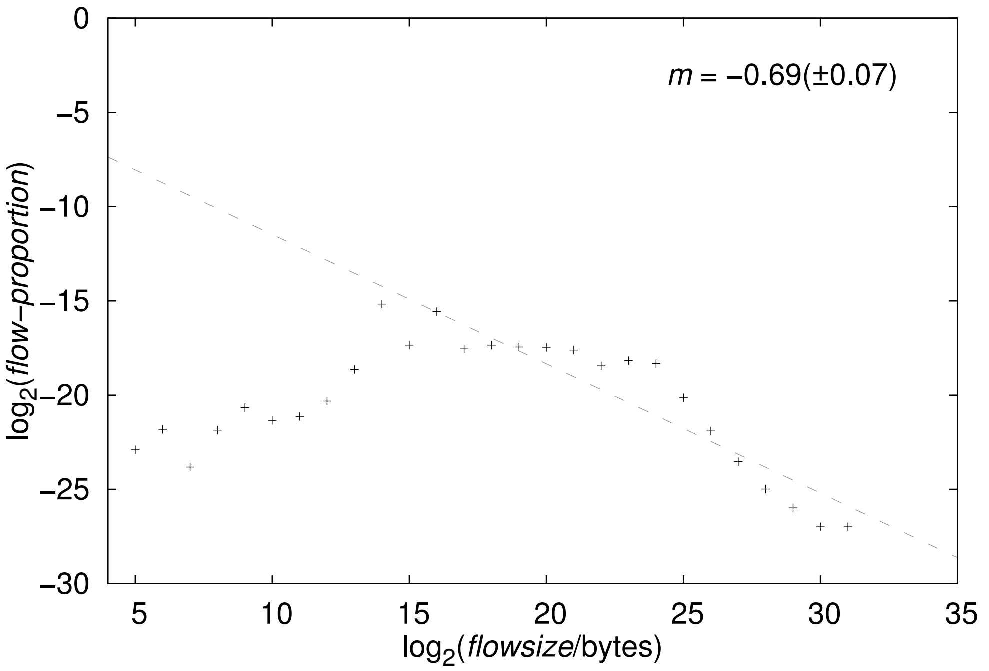


(d) SMTP

Figure 3.3: Logarithmic binned flow size distributions for selected application protocols. (cont.)

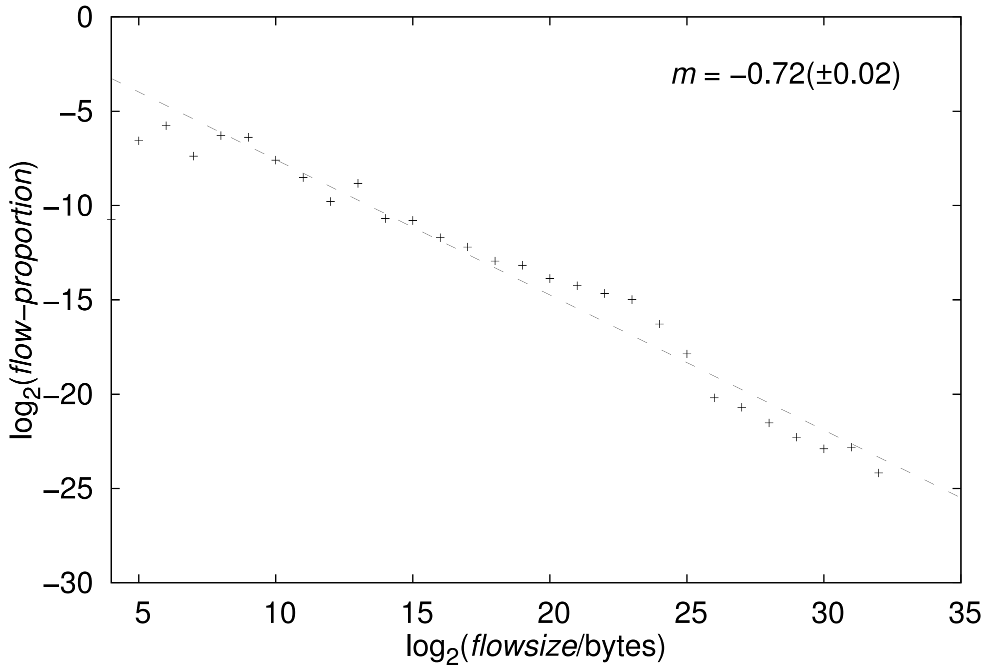


(e) SSH

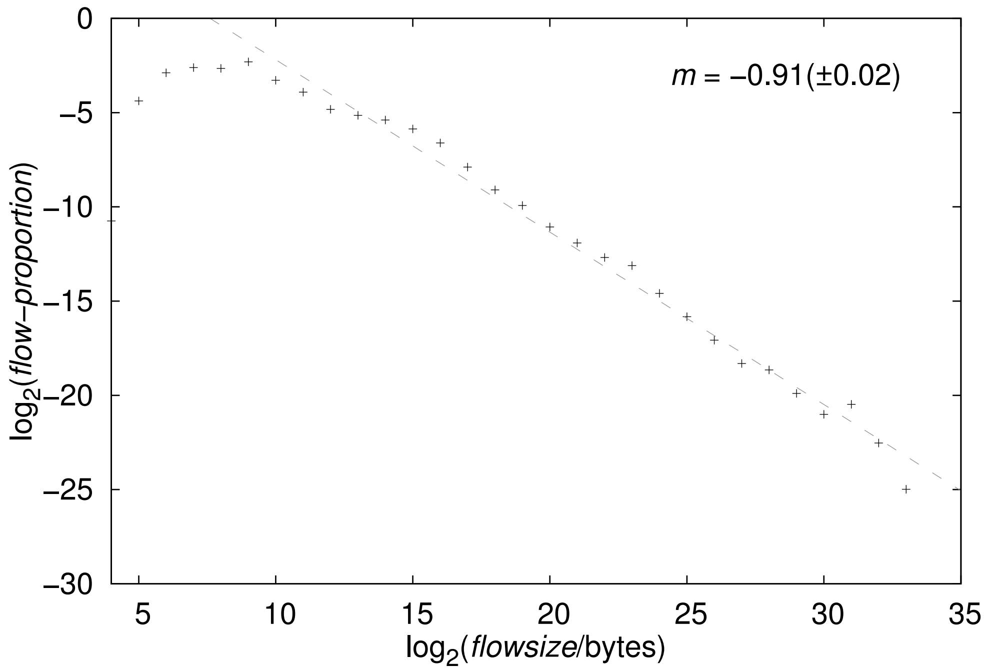


(f) OpenVPN

Figure 3.3: Logarithmic binned flow size distributions for selected application protocols. (cont.)



(g) other



(h) all

Figure 3.3: Logarithmic binned flow size distributions for selected application protocols. (cont.)

3.4.2.2 Results

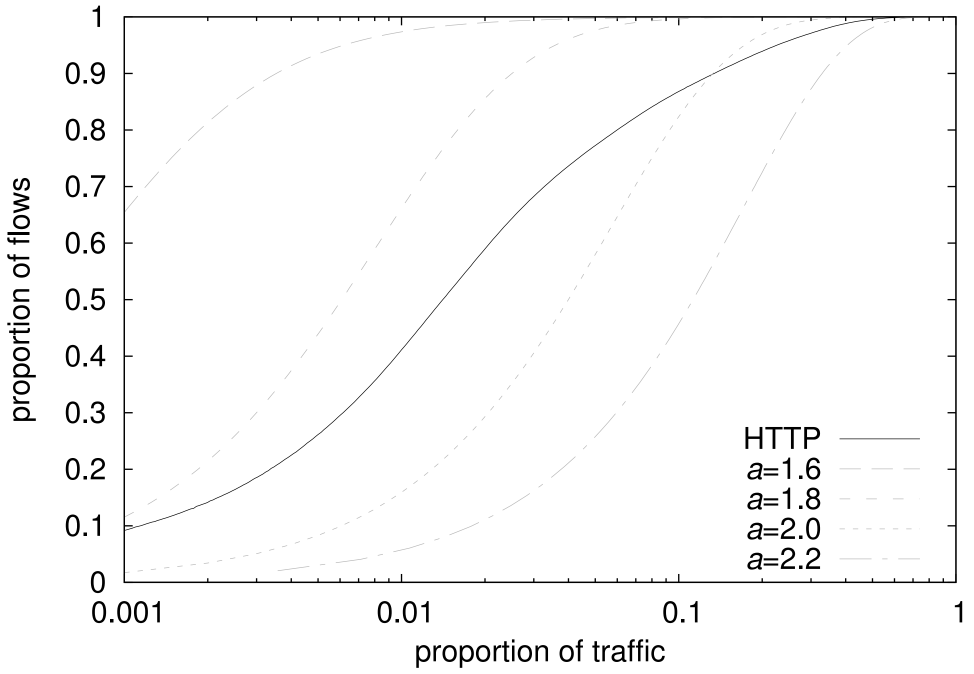
In Figure 3.3, the results of the logarithmic binning are shown together with the fitted lines of the linear regression. Most of the distributions have a nice linear part spanning over a wide range of flow sizes, making them mostly Pareto distributed. Nevertheless, there is always an increasing part at small flow sizes, up to a flow size of about 2^8 to 2^{10} bytes. Therefore, not all values were used for the linear regression. Instead, only the maximum value and values of larger flow sizes were used, respecting the fact that this is about the point where the linearity starts.

The approximated exponents $a = 1 - m$ reach from 1.53 to 2.13. It is noticeable that the protocols HTTP, DNS and SMTP, which have exponents close to 2, are also the protocols which hold most of the flows of the overall traffic (see Table 3.1). Furthermore, the protocols FTP, SSH and OpenVPN, which have exponents around 1.6, are known to be used for long lasting sessions and/or tunnels, which – once established – are likely to persist for a longer time. This again explains, why a is lower in relation to the other protocols. Noticeable as well is that for SSH – partly running on the standard port – there is a large amount of flows smaller than 2^{13} bytes, which could result from vulnerability attacks, while for OpenVPN – running on a secret port – there are very few flows smaller than 2^{13} bytes.

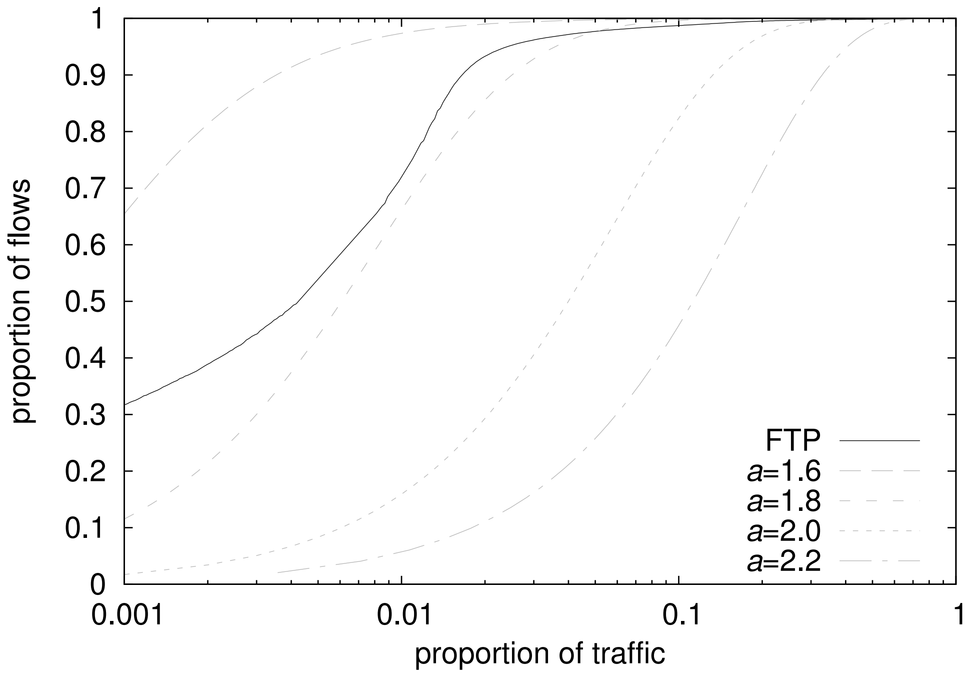
The overall traffic is expected to be mostly Pareto distributed as well, similar to the protocols contributing most (HTTP, DNS, SMTP). This corresponds to Figure 3.3(h).

3.4.3 Mouse Trapping efficiency

In Figures 3.4 and 3.5, the relation between the normalized flow and traffic CDFs of the actual data is shown. Although these gradients cannot be expected to look exactly like the theoretical ones because of the »slow start« and other »non-linearities« of the real flow size distributions, the similarities to Figure 3.1 are obvious nevertheless. If you look at the slope of DNS for instance, which has the most Pareto-like distribution with $a = 2.13$, you see that it is very similar to the

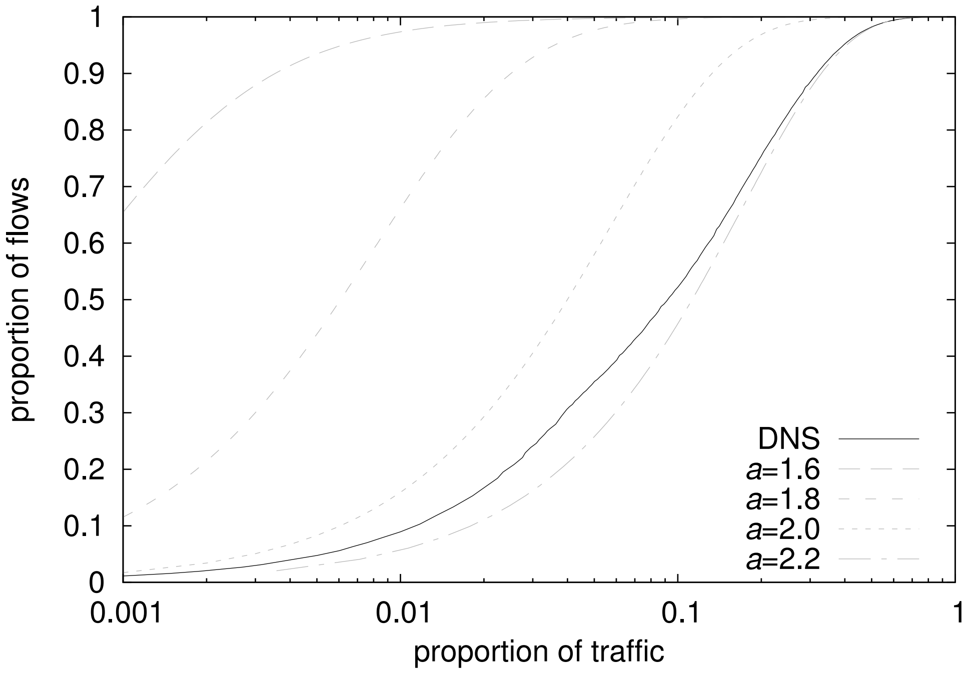


(a) HTTP

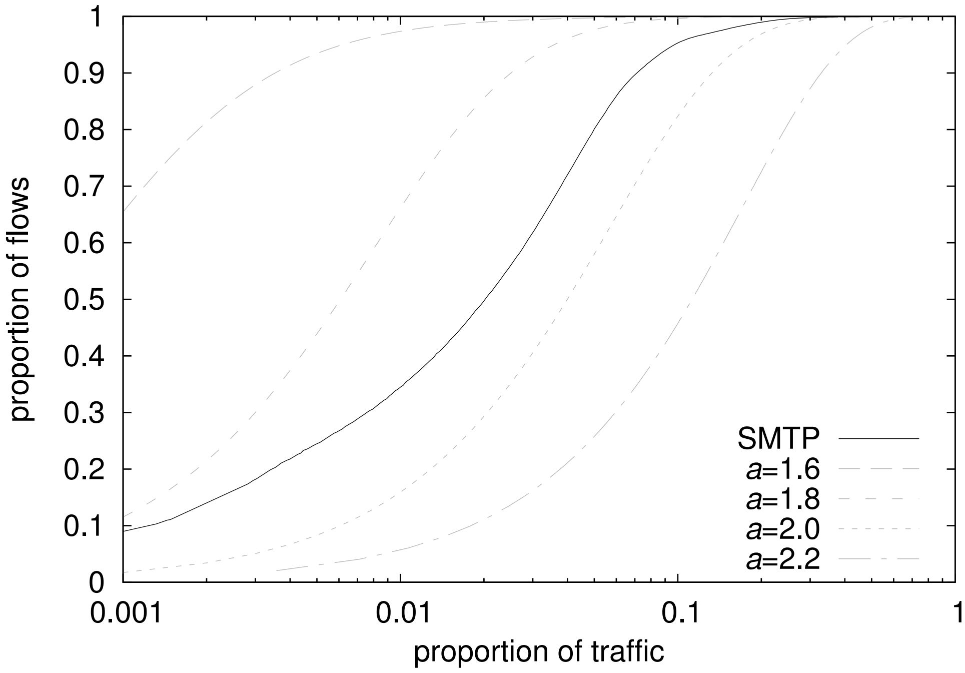


(b) FTP

Figure 3.4: Relation between cumulated flows and cumulated traffic for selected application protocols.

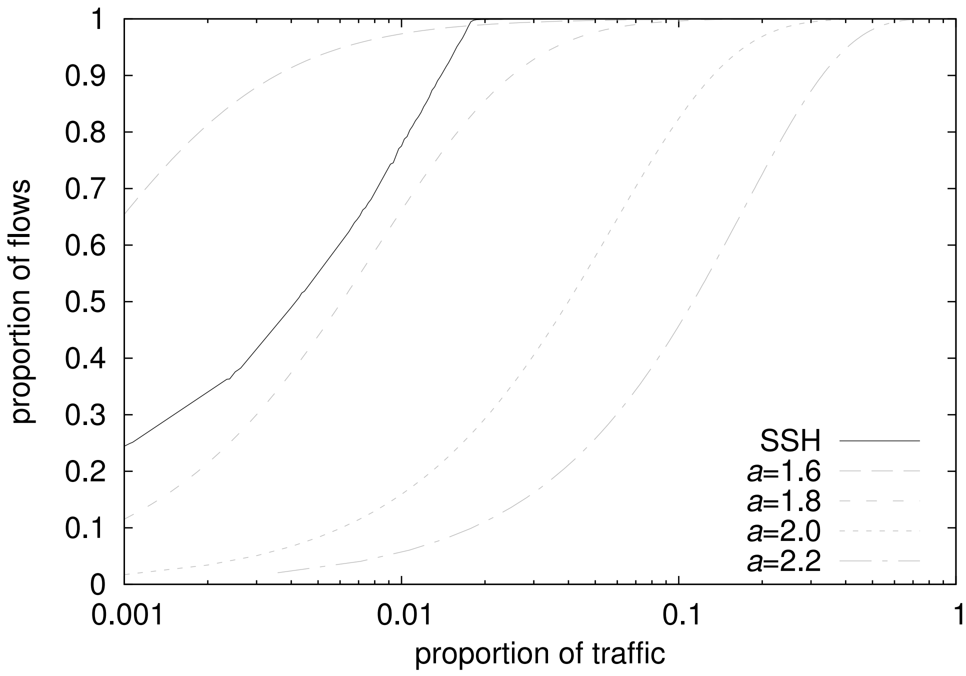


(c) DNS

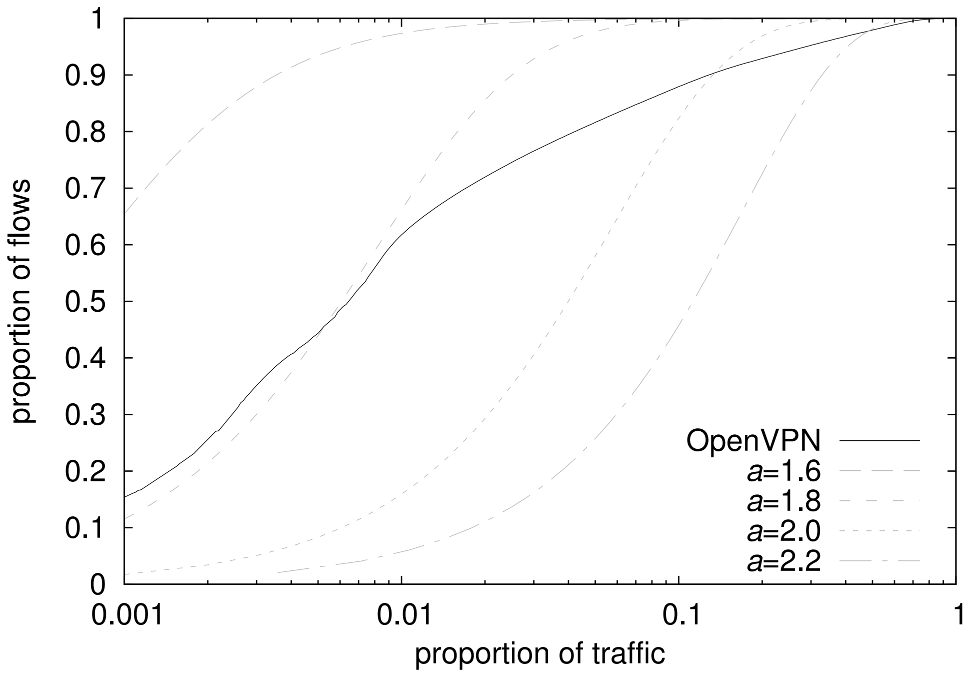


(d) SMTP

Figure 3.4: Relation between cumulated flows and cumulated traffic for selected application protocols. (cont.)

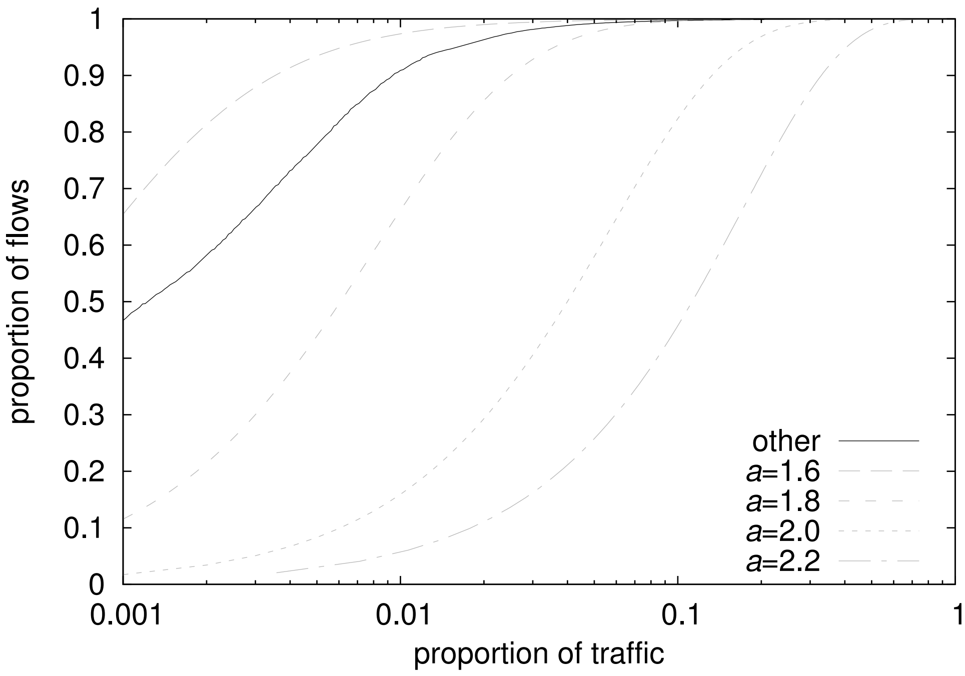


(e) SSH

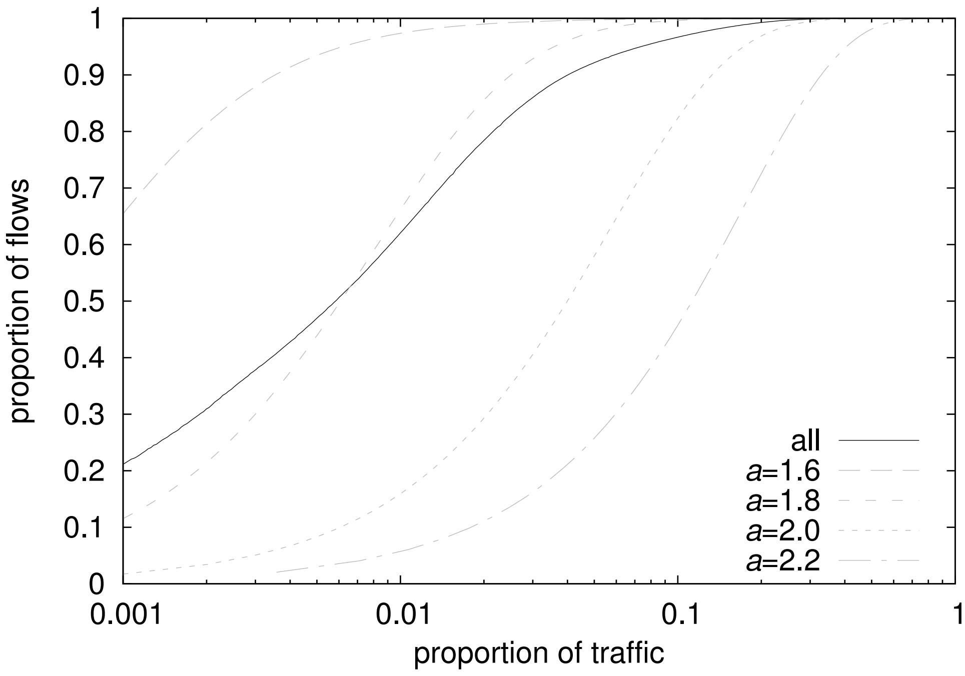


(f) OpenVPN

Figure 3.4: Relation between cumulated flows and cumulated traffic for selected application protocols. (cont.)



(g) other



(h) all

Figure 3.4: Relation between cumulated flows and cumulated traffic for selected application protocols. (cont.)

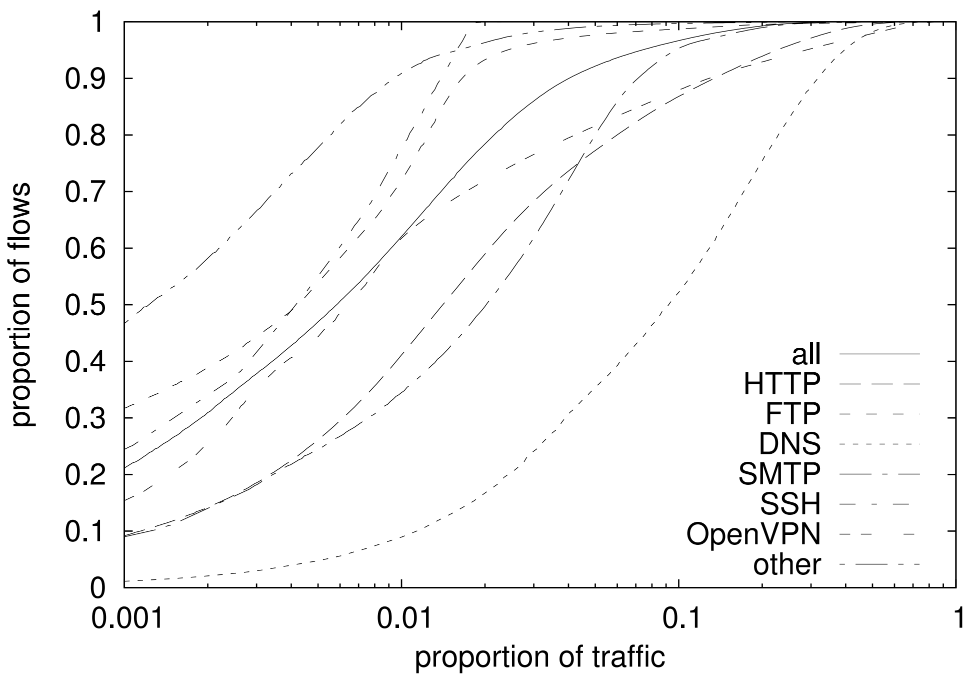


Figure 3.5: Relation between cumulated flows and cumulated traffic for all ports in comparison.

theoretical slope for $a = 2.2$. Compared to that, FTP, with a measured $a = 1.66$, is between the theoretical slopes for $a = 1.6$ and $a = 1.8$, as expected. The effect of the large amount of small flows in the SSH distribution is that the slope reaches almost 100% of the flows already at 2% of the traffic. On the other hand, for OpenVPN, which has very few small flows, the slope is the most shallow one. In this graph you can already see, which reduction rate r you can expect for a given loss rate l and vice versa by using the proposed reduction method.

3.4.3.1 Reduction rates

In Table 3.2, the reduction rates for the measured data are shown for three different given loss rates. The values are the mean values and standard deviations of 65 days, if the reduction is done on the data of each day separately. If information about only 1% of the traffic may be sacrificed, only about half of the flow records can be removed. But already at a loss rate of 5% the overall flows can be reduced by about

Protocol	r (l=10%)	r (l=5%)	r (l=1%)
all	96.6 (±1.1)	91.8 (±2.7)	62.0 (±6.0)
HTTP	86.7 (±1.7)	77.2 (±2.0)	42.3 (±3.7)
FTP	83.7 (±27.1)	79.2 (±29.4)	49.4 (±30.3)
SMTP	86.9 (±15.2)	73.6 (±19.4)	32.0 (±11.0)
DNS	52.5 (±4.8)	35.3 (±3.6)	8.5 (±1.3)
SSH	99.9 (±0.3)	98.6 (±6.5)	76.1 (±23.6)
OpenVPN	80.7 (±11.3)	71.3 (±14.8)	47.3 (±18.9)
other	99.2 (±1.2)	97.7 (±3.0)	84.6 (±10.7)

Table 3.2: Reduction rates for different given loss rates.

90%, achieving a reduction ratio of 1:10. For the important protocol HTTP, the reduction rate is at about 80% (1:5). Only DNS seems to be a problem. Although it is the protocol with the most Pareto-like distribution, the reduction rates are weak because of the large exponent ($a = 2.13$) and the – compared to the other protocols – small flow size bandwidth of only 2^{19}. Therefore, in most cases it will be not advisable to select DNS as a subset with an independent threshold, but instead to let it in the »others« group and accept that due to a higher threshold more information about DNS will be erased.

3.5 Chapter summary

In this chapter we introduced the flow data reduction scheme *Mouse Trapping*. We showed by simulation that *Mouse Trapping* is an effective method to reduce the amount of flow data by only keeping the large flows, which represent the main part of the overall traffic. The theoretical simulation only assumed a power-law like flow size distribution and no other characteristics of the flow data. The analysis per application protocol showed that some subsets of traffic exist, for which this assumption is not true. So in the unusual cases that the flow sizes are not power-law distributed at all, the method does not work very well. Also this is a lossy reduction method and might not be suitable for some applications which depend on the information of the small flows, like, for example, some subtle intrusion-detection-system. But our data analysis and other studies showed that common internet traffic is almost power-law distributed and it can be assumed that this method works well in general. The evaluation showed that for common traffic a data reduction of up to 90% can be achieved with a loss of information about only 5% of the traffic. Since for many applications the information loss about the small flows is acceptable, it can be a useful method, for example, to condense old data in flow data repositories or to use it in mediators to filter data that is forwarded to central collectors.

The future work on this approach may include research on effective and fast methods to determinate the threshold without the need to scan through all the traffic data, by applying sampling techniques, for instance. This is especially interesting for the application in mediators, that need a fast decision, if a flow should be forwarded to a Collector. Another important field as well is the evaluation of Mouse Trapping for traffic data measured by packet and flow sampling mechanisms, which are already widespread especially in high-speed backbone measurements.

Chapter 4

Interactive Flow Data Analysis: FloX (Flow eXplorer)

4.1 Motivation

A major challenge in flow based traffic measurements is the extraction of important information, its presentation and the interaction of the observer with the data. There are generally way too many raw flows to get an overview just by inspecting the flow tables themselves. Especially for flows with many flow keys the mutual interrelation remains hidden for the human eye. To efficiently work with such data it is inevitable to have good analysis and presentation tools, which enable the administrator to extract necessary and important information. Of course, there cannot be a single »one-fits-all« tool, and the design highly depends on the requirements and applications for which such a tool is going to be used. These tools or user interfaces might range from simple-to-observe »OK/not-OK« status displays that can report the existence of a problem at a glance without giving any further detail, to complex assistance tools that aid an investigator in actively examining the data in a structured and ingestible way.

In this chapter we will describe the design and implementation of a simple but very useful tool called FloX *(Flow eXplorer)* [FloX] that is mainly intended to examine the flows of high-bandwidth events by providing rankings of dynamically defined flow aggregates.

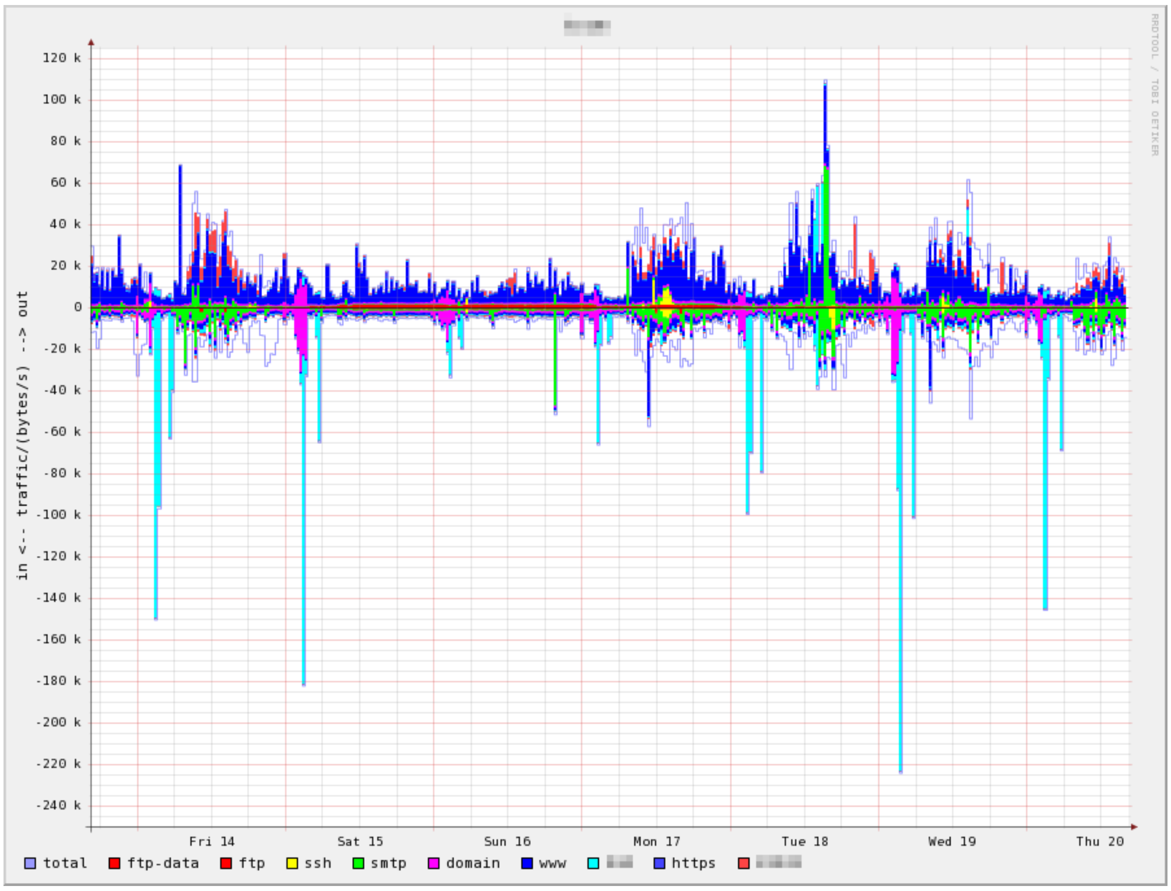


Figure 4.1: Example of separate time series measurements stacked in a single plot, in this case several interesting port numbers.

4.2 Analysis of Peak Events

Many problematic events in network traffic monitoring involve a high bandwidth usage, since this usually results in higher costs, higher latency and worse Quality-of-Service in general. Besides these direct impairments, an unusual high bandwidth usage can be an indicator that something undesired is happening that might compromise the security of the connected systems and infrastructure. Therefore, if in a time series of measured bandwidth usage an unexpected peak appears, it is the usual case that the network administrator would like to know where this peak is coming from, and how exactly the traffic looks like that caused the peak.

A first step would be to measure separate time series for certain subclasses of traffic, like for each of a number of important hosts or interesting port numbers, as in the example shown in figure 4.1. However, the additional information is very limited and such an approach is not flexible, since it has be known in advance which subclasses of traffic might be interesting. The investigator might also want to analyze the characteristics of the peak according to further attributes such as the client IP addresses, used transport protocols, or the distribution of port number usage in the higher port ranges above the »well known ports«. The latter is very important for identifying certain malware activities, for instance.

If in addition to a simple time series fine grained flows are measured as well, all interesting information about the peak is stored somewhere in flow record tables, but is unaccessible for a human without further analysis tools. So if the investigator is given easy access to all this information, it would be possible to answer questions such as:

- Where did the traffic come from and where did it got to?
- What does the host relation matrix of the traffic look like?
- Are only two hosts involved? (1:1 relation)
- Is one host addressed by many distributed hosts, like it is the case for DDOS attacks? (N:1)
- Is one host addressing many distributed hosts, like it is the case for suddenly demanded services, but also for hosts that scan for other hosts. (1:N relation)
- Is the event not related to a single host, but contains arbitrary host relations, like it is the case for temporary rerouting of traffic in carrier network. (N:M relation)
- What services is the event related to? WWW, email, file-sharing...
- Is the event due to a single large flow (a single TCP connection for instance) or many small flows?
- Is it TCP or UDP traffic?

Hence, there is the need for a tool that provides a way to »dig« into a high traffic event in a flexible way and to surface as much information about the characteristics of the responsible traffic as possible and necessary. It should make use of the fact that in the given scenario of a sudden bandwidth peak, the additional traffic accounts for a big share of the overall traffic.

4.3 Divide and Conquer Approach

The basic idea derives from typical »divide-and-conquer« approaches, for example the one a system administrator might take, when the hard disk space usage has suddenly increased unexpectedly or a filesystem is even completely full and he or she wants to find out what the reason for this increase is, that is, which directories or files are responsible for the increase.

A simple method is to start at the root of the affected file system hierarchy and sum up the size of each directory. In Unix environments this is usually done with a simple '`du -s *`' command. The experienced administrator will notice unusual sizes – also knowing in which directories there usually is writing activity – or automated scripts can compare the sizes with former sizes. Hereupon the administrator (or the script) enters the conspicuous directories – in most cases only one – and again sums up the sizes of the subdirectories. This process is repeated until as few directories or files as possible are identified, which are responsible for the increased disk space usage.

To characterize this in a more abstract way: the space of data elements is divided into subspaces according to a stratification criterion. If possible, this criterion is ideally chosen in a way that supposedly most of the unusual data elements are contained in one or few of the subspaces. (In a file system hierarchy there is no such choice, of course.) Then the sizes of all elements are summed up for each of the subspaces. If one of the sums is unusually high, this subspace is chosen as a new basis and a new stratification criterion is chosen. This is repeated until a minimal subspace is found containing all or most of the unusual data elements.

Following this concept we designed a simple, yet very useful and flexible tool and built a proof-of-concept implementation. This tool helps to explore the internal

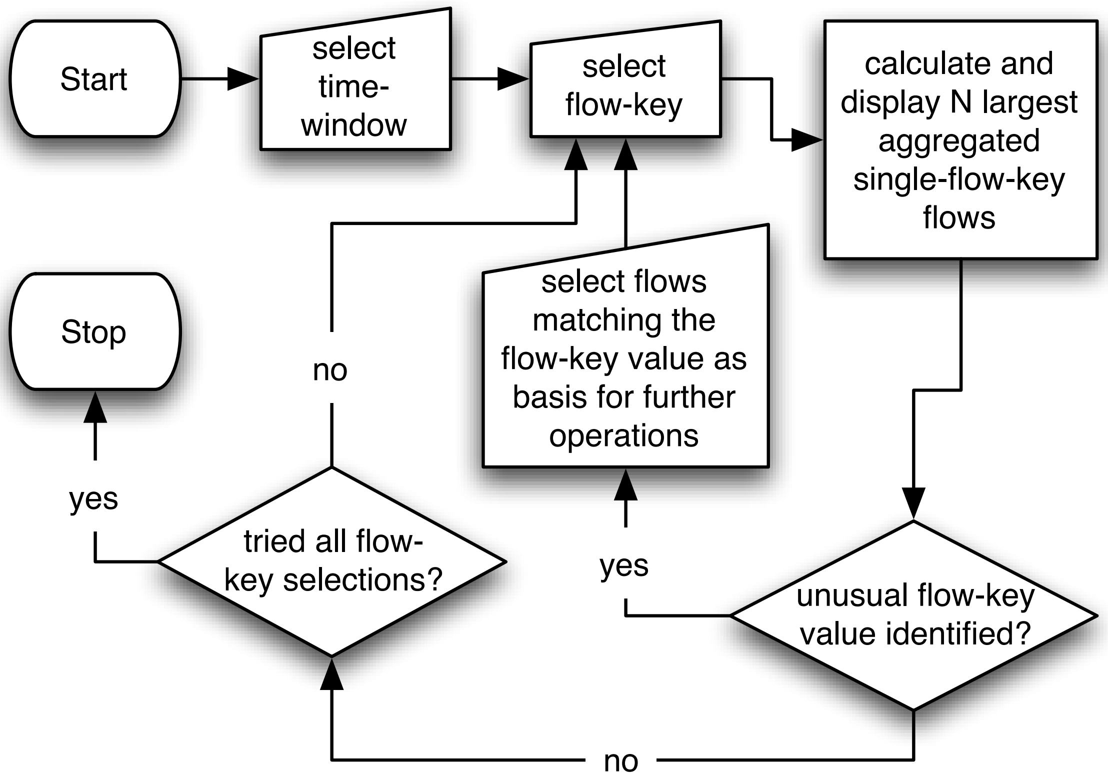

Start
select time-window
select flow-key
calculate and display N largest aggregated single-flow-key flows
Stop
no
select flows matching the flow-key value as basis for further operations
yes
tried all flow-key selections?
yes
unusual flow-key value identified?
no

Figure 4.2: Diagram of the work flow for analyzing flow records of a peak event with the FloX software.

structure of a large number of flow records. Therefore the name FloX has been chosen, which derives from *Flow eXplorer*.

The diagram in figure 4.2 illustrates the general work flow for using FloX to analyze a traffic peak event. The peak event has to be defined by selecting an appropriate time-window that should not contain much more than the time period of unusual high traffic. This information usually can be gathered from a general bandwidth usage graph. Any of the flow-keys can be chosen as the stratification criterion. Then the sizes of all flows are summed up for each existing value of the chosen flow key. Technically this is the same as aggregating the flow records to simple »single-flow-key flows« containing only the chosen one. The *N* largest of these aggregated flows are then displayed to give an overview which values of the chosen flow-key are the most frequent ones in the observed traffic. It is important to note that flows can have several counters that could equally serve as »size« of the flow, like the number of bytes or the number of packets. Therefore it is also necessary to select one of the counter fields as the ranking criterion.

For a typical peak event, the peak is significantly higher than the average traffic. So, if a suitable flow-key has been selected, the largest aggregated flow should stand out of the rest or at least an unusual value should be found among the largest aggregates, like unexpected port numbers or IP addresses. Otherwise a different flow-key can be selected for a new aggregation. After an interesting value has been identified, this value can be selected and all further operations are only done on the subspace of flow records matching the chosen flow-key value. Then another flow-key is selected for aggregation and the whole procedure is repeated until no further flow-key can be selected in order to separate the unusual traffic from the rest. At this point, as many flow-key values as possible have been identified defining the characteristics of the peak traffic.

4.4 Implementation

FloX is a web application and has been implemented in the PHP scripting language. It presents an HTML based user interface, and the interaction is realized with links and HTML forms that iteratively re-invoke the same script again. Since the PHP script itself is »stateless« (a better expression might be »single-state«), the state of the analysis process is stored in the parameters of the PHP script URL, a very common technique for simple web applications.

Besides managing the user interaction, the main task of the script is to translate the user requests into SQL queries and to send them to a database. The performance of the script itself is not critical, because all expensive calculations are processed by the database management system. Hence, the performance of the database determines the performance of FloX, and good performance adjustments of the DBMS are necessary. FloX has been tested with PostgreSQL and MySQL, but since it uses a database-independent SQL-API for PHP, *Pear-DB*, the adaption to other supported DBMS should be easy.

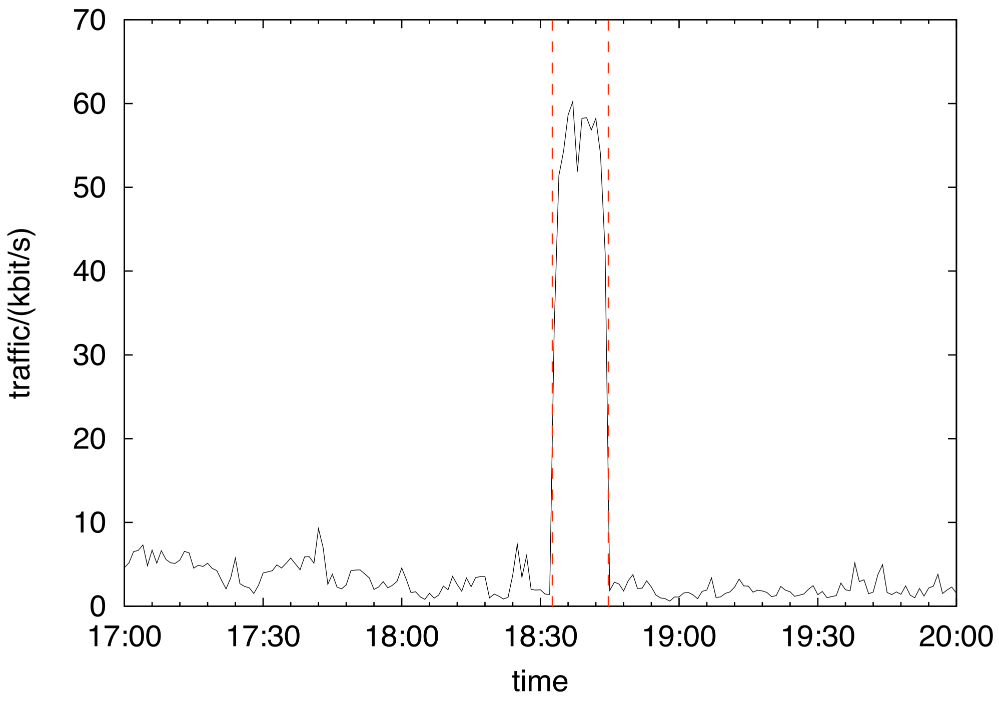
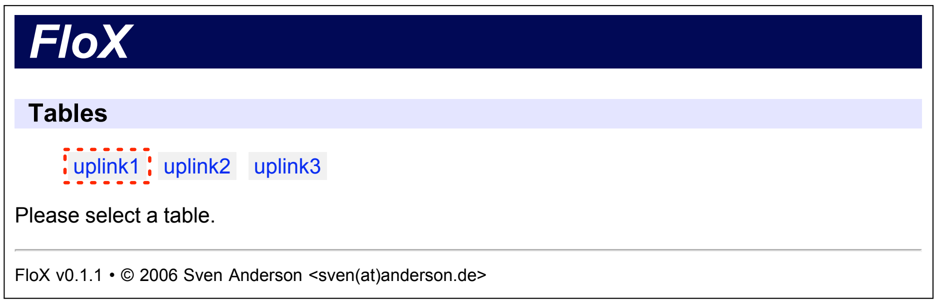


Figure 4.3: An exemplary peak in a traffic usage that will be analyzed with FloX.

The configuration of FloX is simple but flexible and can be adapted to many situations. First of all, the connection to the database must be configured. Although FloX has been designed to work with flow record tables created by the traffic measurement software pmacct [pmacct], it should be able to handle virtually any kind of flow record table. It only has to be defined in the configuration file which columns in the table represent counter fields, which column should be used as the time stamp and which format the time stamp has. All columns/fields that are not counter fields are handled as flow keys. This is true even for the time stamp field, which is strictly speaking not a real flow key, but it can come in handy in certain situations.

FloX has been published as Open Source Software under the GNU General Public License [GPL] and is offered as a free Internet download [FloX].



Please select a table.

FloX v0.1.1 • © 2006 Sven Anderson <sven(at)anderson.de>

Figure 4.4: Select database table in startup screen.

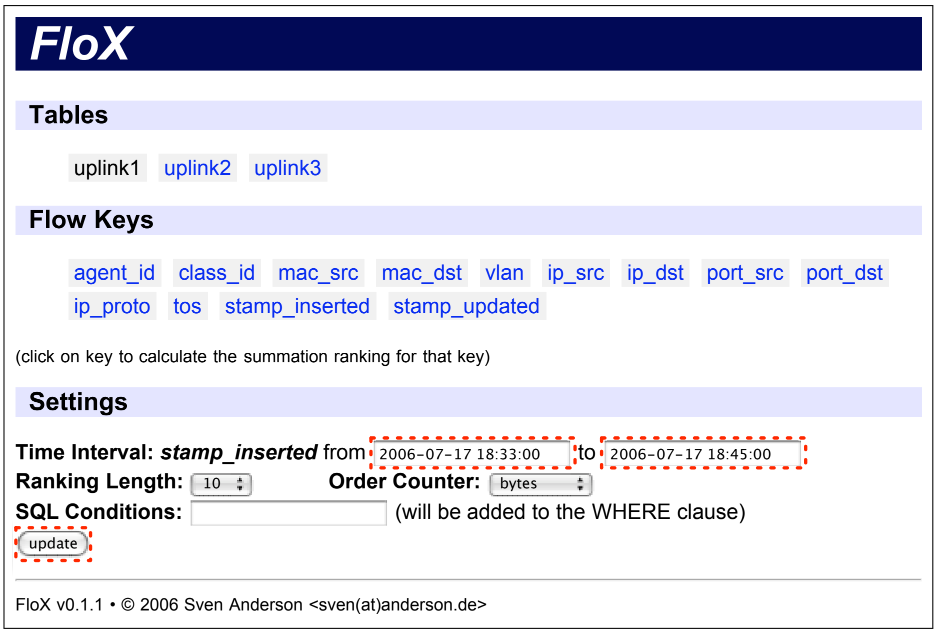
4.5 Usage Example

In this section we will show with an example from real traffic data how a peak event can be analyzed with FloX. Figure 4.3 shows a plot of the network traffic over time. These types of plots are very common in network management and are usually generated with time series tools such as RRDTool. However, no matter if they represent the broad traffic of a whole traffic exchange point, a certain network link or a very specific subset of traffic like for a single host or port, the information about the peak is very limited. If a fine grained flow measurement has been done in parallel to the time series, there is much detailed information hidden about the inner structure of such a peak event.

The flow records used for this example are very fine grained. They have a time resolution of one minute, eleven different flow keys and the three counter fields for the number of bytes, packets and sessions (TCP connections, for instance).

As a first step, it is necessary to select the smallest window that contains the whole peak. Since the flow data for this example has a time resolution of one minute, the time window ranges from 18:33h to 18:45h (see marks in figure 4.3).

In figure 4.4, the startup screen of FloX is shown. If the traffic data has been separated into different tables, for example for different network links, a table has to be selected.


FloX

Tables

uplink1 uplink2 uplink3

Flow Keys

agent_id class_id mac_src mac_dst vlan ip_src ip_dst port_src port_dst ip_proto tos stamp_inserted stamp_updated

(click on key to calculate the summation ranking for that key)

Settings

Time Interval: ***stamp_inserted*** from 2006-07-17 18:33:00 to 2006-07-17 18:45:00
Ranking Length: 10 **Order Counter:** bytes
SQL Conditions: (will be added to the WHERE clause)
update

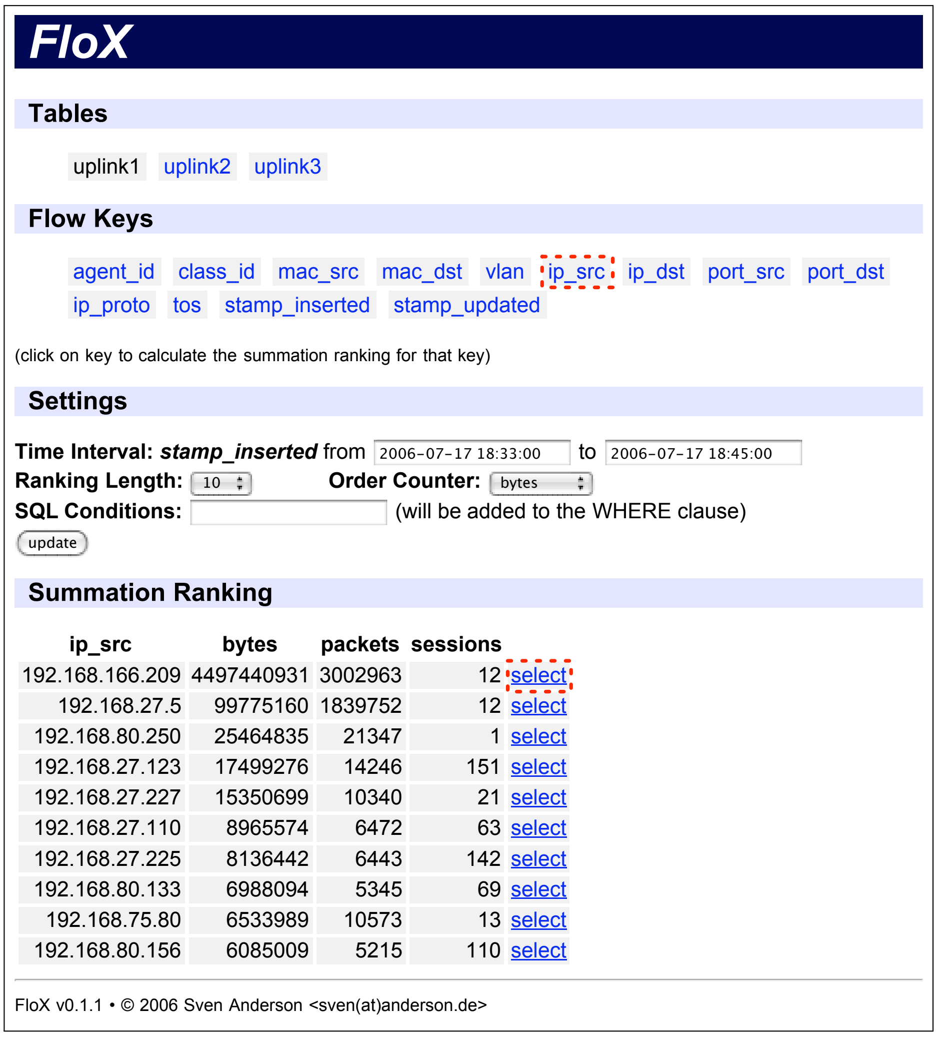
FloX v0.1.1 • © 2006 Sven Anderson <sven(at)anderson.de>

Figure 4.5: Enter time window of examination.

In the next step (figure 4.5), the time window for examination has to be entered which is going to be examined, that is, only flow records with a matching time stamp are used for analysis. In this case, this is the time window of the peak as determined from the time series plot before. Since the implementation of the user interface is purely based on passive HTML code without any client executed code like Javascript, the »update« button has to be pressed in order to store that input.

Now the first flow key must be chosen, for which the flow counters are summed up for each of its values (see figure 4.6). In this example we chose the source IP address (*ip_src*) as the first selection in order to find out where most of the flows came from, because there is a good chance that most of it came from one IP. The calculation is started by clicking on the according flow key. After the calculation has finished, the result is shown in the *Summation Ranking* section.

As expected, the vast majority of the traffic came from a single IP address (192.168.166.209). However, in the case of a DDOS attack this would be different and therefore can at this point already be excluded as a possible reason for the traffic peak. Also the relatively low total number of sessions for this IP address



Tables

uplink1 uplink2 uplink3

Flow Keys

agent_id class_id mac_src mac_dst vlan ip_src ip_dst port_src port_dst ip_proto tos stamp_inserted stamp_updated

(click on key to calculate the summation ranking for that key)

Settings

Time Interval: ***stamp_inserted*** from 2006-07-17 18:33:00 to 2006-07-17 18:45:00
Ranking Length: 10 **Order Counter:** bytes
SQL Conditions: (will be added to the WHERE clause)

Summation Ranking

ip_src	bytes	packets	sessions	
192.168.166.209	4497440931	3002963	12	select
192.168.27.5	99775160	1839752	12	select
192.168.80.250	25464835	21347	1	select
192.168.27.123	17499276	14246	151	select
192.168.27.227	15350699	10340	21	select
192.168.27.110	8965574	6472	63	select
192.168.27.225	8136442	6443	142	select
192.168.80.133	6988094	5345	69	select
192.168.75.80	6533989	10573	13	select
192.168.80.156	6085009	5215	110	select

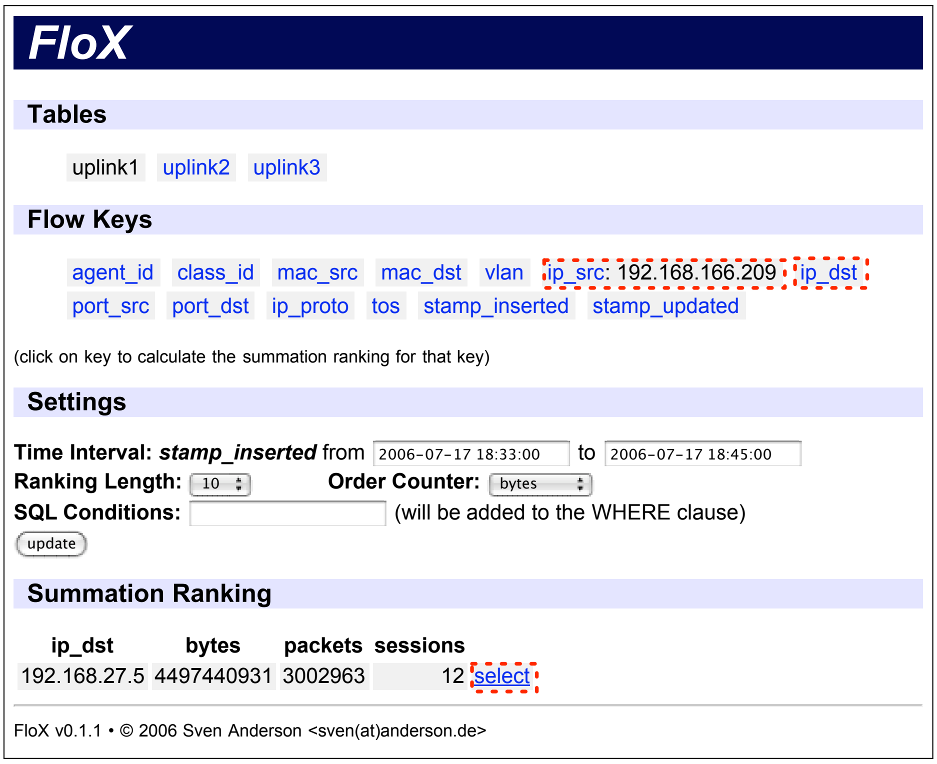
FloX v0.1.1 • © 2006 Sven Anderson <sven(at)anderson.de>

Figure 4.6: Aggregated flows for each source IP address.

already reveal something about the type of traffic. It obviously consists of a few long lasting sessions that transferred a large amount of data, like it is typical for file transfers or media streams, for instance. Accordingly, there cannot be a lot different destinations for these packets.

Since the traffic responsible for the peak is obviously connected to this single IP source address, it can be selected as a fixed value for the flow key *ip_src* by pressing »select« next to it. This means that all further operations will only be made with the flows that match that specific source IP address.

In figure 4.7 the next step is shown. As an indicator that a fixed flow key value


FloX

Tables

uplink1 uplink2 uplink3

Flow Keys

agent_id class_id mac_src mac_dst vlan ip_src: 192.168.166.209 ip_dst
port_src port_dst ip_proto tos stamp_inserted stamp_updated

(click on key to calculate the summation ranking for that key)

Settings

Time Interval: ***stamp_inserted*** from 2006-07-17 18:33:00 to 2006-07-17 18:45:00
Ranking Length: 10 Order Counter: bytes
SQL Conditions: (will be added to the WHERE clause)
update

Summation Ranking

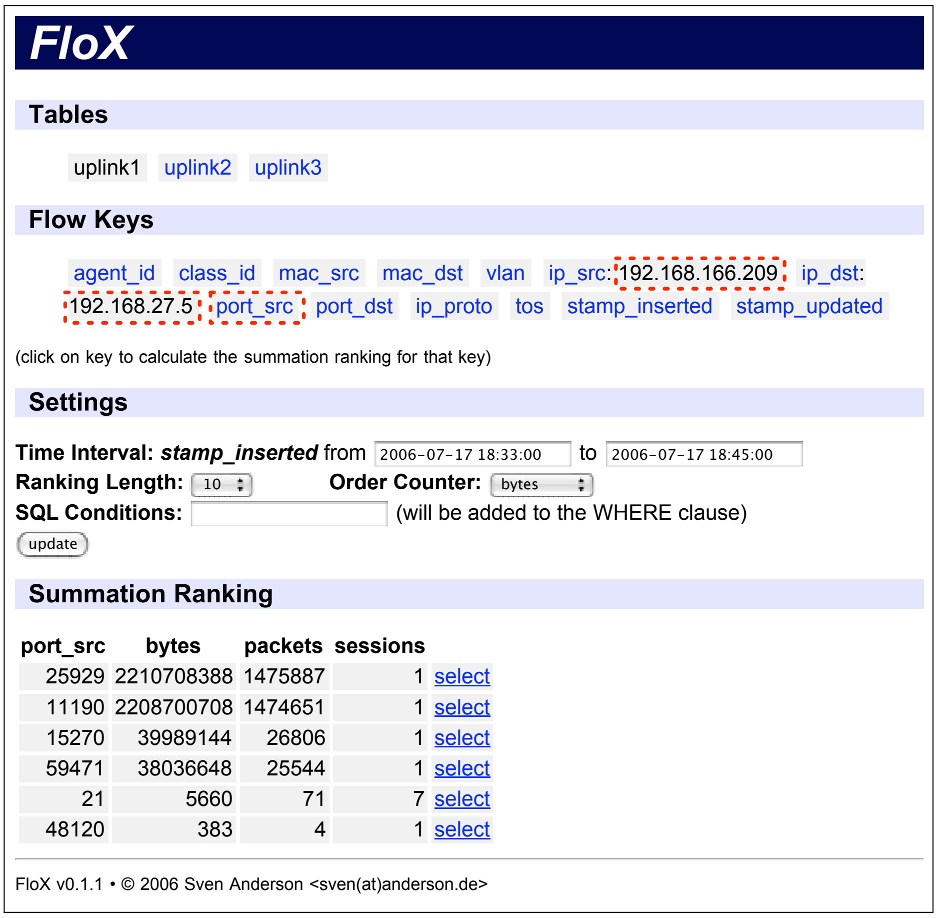
ip_dst	bytes	packets	sessions	
192.168.27.5	4497440931	3002963	12	select

FloX v0.1.1 • © 2006 Sven Anderson <sven(at)anderson.de>

Figure 4.7: Aggregated flows for each destination IP address with a selected source IP address.

has been selected and all operations only apply to a subset of flows, the value is shown next to the flow key, in this case *ip_src*. Now the destination IP address is chosen by clicking on the *ip_dst* flow key. Again, after calculation the result is shown in the *Summation Ranking*. As predicted, there are not a lot of different destinations. In fact all traffic went to a single destination. Although technically not necessary, since it does not further reduce the flow subspace, we again select the destination IP address as a fixed value to retain that property of the peak traffic.

In figure 4.8 it can be seen that again the fixed values are shown next to the flow key names. Now that we know that the peak was caused by 1:1 traffic between just two hosts, it is interesting to find out what kind of traffic it was. Therefore the source port flow key is selected by clicking on *port_src*. This time the result shows that the traffic is not bound to a single value but is distributed on packets mainly coming from two ports. These ports have arbitrary high numbers, therefore no


FloX
Tables
uplink1
uplink2
uplink3
Flow Keys
agent_id
class_id
mac_src
mac_dst
vlan
ip_src: 192.168.166.209
ip_dst: 192.168.27.5
port_src
port_dst
ip_proto
tos
stamp_inserted
stamp_updated
(click on key to calculate the summation ranking for that key)
Settings
Time Interval: stamp_inserted from 2006-07-17 18:33:00 to 2006-07-17 18:45:00
Ranking Length: 10
Order Counter: bytes
SQL Conditions: (will be added to the WHERE clause)
update
Summation Ranking

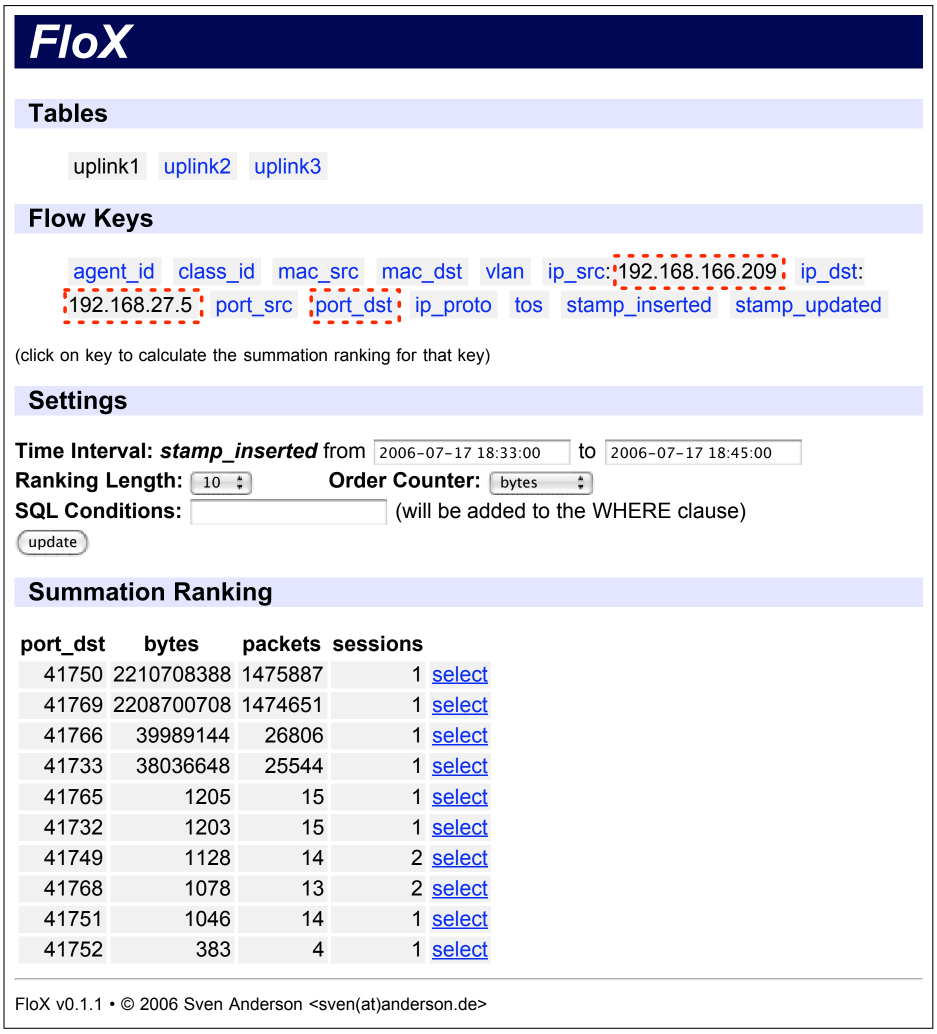
port_src	bytes	packets	sessions	
25929	2210708388	1475887	1	select
11190	2208700708	1474651	1	select
15270	39989144	26806	1	select
59471	38036648	25544	1	select
21	5660	71	7	select
48120	383	4	1	select

FloX v0.1.1 • © 2006 Sven Anderson <sven(at)anderson.de>

Figure 4.8: Aggregated flows for each source port with a selected destination and source IP address.

conclusions can be drawn from them. However, in the results a small amount of traffic coming from port 21 can be seen. This gives an important hint, since 21 is the port number for the control plane communication of the *File Transfer Protocol (FTP)* [RFC 959]. Unlike many other protocols, for FTP the control and user data planes are separated into different TCP connections. Therefore the actual file transfers happen over connections with arbitrary ports that are initially negotiated in the control plane.

At this point we know already much about the origin of the peak: an FTP client at IP address 192.168.27.5 transferred four files from an FTP server at IP address 192.168.166.209. Two files were about 2.2 GB and two about 40 MB in size. The rest of the connections are either directory lists or small files. This is confirmed by the re-



FloX

Tables

uplink1 uplink2 uplink3

Flow Keys

agent_id class_id mac_src mac_dst vlan ip_src:192.168.166.209 ip_dst:192.168.27.5 port_src port_dst ip_proto tos stamp_inserted stamp_updated

(click on key to calculate the summation ranking for that key)

Settings

Time Interval: ***stamp_inserted*** from 2006-07-17 18:33:00 to 2006-07-17 18:45:00
Ranking Length: 10 **Order Counter:** bytes
SQL Conditions: (will be added to the WHERE clause)

Summation Ranking

port_dst	bytes	packets	sessions	
41750	2210708388	1475887	1	select
41769	2208700708	1474651	1	select
41766	39989144	26806	1	select
41733	38036648	25544	1	select
41765	1205	15	1	select
41732	1203	15	1	select
41749	1128	14	2	select
41768	1078	13	2	select
41751	1046	14	1	select
41752	383	4	1	select

FloX v0.1.1 • © 2006 Sven Anderson <sven(at)anderson.de>

Figure 4.9: Aggregated flows for each destination port with a selected destination and source IP address.

sults of figure 4.9, where the distribution of the destination port *port_dst* is analyzed. All ports are close to each other, which is typical for a client program requesting several outgoing TCP sockets from the operating system.

This example shows how it is possible to analyze the nature of a sudden network traffic peak with FloX in a few steps. Of course there are more complicated cases, but the general proceeding stays the same. And even in cases where it is not possible to identify any flow key based properties, this in itself is a possibly important information about the traffic and might allow further conclusions.

4.6 Similar Tools

There are many tools to analyze flow based traffic data, such the popular FlowScan software package of the CAIDA project [Plonka 2000], which makes use of other tools like cflowd and RRDtool. However, they are not interactive, but have to be configured in advance to extract the information of interest from the flow records.

The only freely available tool that has a kind of interactive »drill-down« analysis capability is the Netpy tool [Estan and Magin 2005]. But it uses sampling and its own database and therefore is limited in its applicability.

Besides this, all these tools are specific NetFlow tools and therefore only can handle the flow record formats that NetFlow supports. In contrast, FloX is flexible regarding the defined flow keys and can also handle flow records from flexible template based concepts like IPFIX.

4.7 Chapter Summary

In this chapter we presented a simple but very effective tool for the interactive analysis of flow records. Its main purpose is to »dig« into the components of special traffic events like peaks and surface as much as possible about how the flows look like that are responsible for the main traffic during that event.

For that purpose the user repeatedly chooses flow keys and the most interesting values from top N rankings in an iterative process, so that step by step more properties of the flows causing the event are determined.

We demonstrated with an example how a traffic peak could be traced back in only three iterations to FTP file transfers of four large files between two specific hosts.

Chapter 5

Distributed Monitoring of VoIP Traffic: SIPFIX

5.1 Motivation

The deployment of *Voice-over-IP (VoIP)* telephony is increasing fast. Not only are there more and more telephony service providers operating over the Internet and offering free VoIP-to-VoIP calls and cheap rates to PSTN telephones. The future telephony core networks, also known as *Next-Generation-Networks*, will be IP based as well, for example the mobile phone network defined by the *3rd Generation Partnership Project (3GPP)* including the *IP Multimedia Subsystem (IMS)*. Access providers also started to replace their classic analog and ISDN telephone lines with VoIP based products operating over broadband network technologies such as DSL or cable Internet.

So obviously the need for monitoring solutions for VoIP technologies is increasing as well. Coming from the perspective of a classical circuit-switched telephone network, VoIP brings many new possibilities and improvements, but also a multitude of new risks and challenges. While VoIP itself is part of the application layer it inherits all the properties of the packet switched IP network it is based on, like uncertain bandwidth, variable latency, changing routes and so on. Also has the user direct access to the transport layer, which increases the attack surface of the system security. So, in addition to the classical telephony monitoring needs for billing and the observation of the system-load, performance and operation-faults, there is the need to monitor many other features, which are necessary to measure the Quality-of-Service, verify call integrity or to detect harmful events like billing fraud, Spam-over-IP-Telephony (SPIT), malicious rerouting, interception, manipulation and media injection, to name a few.

Also the borders between VoIP and similar applications like video telephony, media distribution and even instant messaging and email are getting more and more blurred, since the used technologies are constantly extended to support further application types. For example SIP, mainly used for media stream sessions, is being extended to support instant messaging with an extension called SIMPLE, while XMPP, mainly used for instant messaging, is just being extended with Jingle, which manages multimedia sessions in order to support voice and video chat. Obviously the trend goes towards rich signaling protocols, that manage contact relations between users and support real-time and off-line communication in different ways: store and forward (like email), publish-subscribe (presence lists), hop-to-hop (like instant messaging) and setup of direct end-to-end media stream sessions (audio and video telephony, media streaming, white-board). Which paths the different kinds of communication take, whether signaling or content, is difficult to predict and may vary significantly.

This means that monitoring demands get broader and more complex with time, and so do the demands for a versatile and distributed solution able to support all these applications and open to constantly grow with them. But current monitoring schemes are either flexible and distributed ones, but designed for the monitoring of just the transport layer independently of the application layer, or they are application layer specific but static and monolithic, mainly based on APIs or log-file analysis of specific a server software. The latter ones cannot master the complexity of the aforementioned patterns of modern communication systems.

Therefore, a promising approach is to take an existing distributed monitoring scheme for the transport layer and extend it with components that do application layer analysis for specific application protocols and data structures that are able to keep and transport the extracted information. This results in a cross-protocol monitoring system, that is flexible, distributed and scalable in order to scope with the increasing complexity of the communication protocols.

In this chapter, based on the work in [Anderson et al. 2009], we will follow this approach for the widely used *Session-Initiation-Protocol (SIP)*. Because of its broad deployment and since it is part of other standards like the aforementioned IMS of the 3GPP mobile phone network, it can be regarded as the most important among similar protocols. But of course the principles of this concept can easily be adapted to similar protocols like H.323, IAX or XMPP/Jingle.

First we will describe the specific problems of SIP monitoring and the resulting requirements for the monitoring scheme. Then we will shortly introduce IPFIX and the Mediator concept the scheme is based on. We will define new IPFIX Information Elements and Flow Types, which build the core of SIPFIX, describe the necessary and optional device extensions, and recommend the use of certain IPFIX optimizations. In the end we will present a series of use case examples and address some implementation issues.

5.2 Problem Description

Monitoring the various features of SIP traffic and its corresponding media streams as well as measuring related performance metrics bring up several challenges.

SIP is foremost a signaling protocol, and as such organizes the session of a number of media streams. Contrary to these streams, the SIP communication normally does not happen directly between the peers that want to communicate, but is relayed over SIP servers, Session Border Controllers and Proxies, and can be redirected many times before reaching the peer. As the media streams usually do not take the same paths as the SIP packets, one can not assume that there is a single measuring point, where all the traffic belonging to one session can be seen. Therefore a monitoring scheme for SIP has to support the possibility to do measurements at different locations and correlate these measurements.

Another problem is that – similar to data connections of the FTP protocol – the format of the media streams are not SIP specific at all, but usually general RTP streams, for instance. It is therefore impossible to tell just from the media stream to which SIP session it belongs to, not even if it belongs to a SIP session at all.

Accordingly, to identify them as part of a SIP session and relate them to the correct one is only possible, if the session description is extracted from the content of the SIP packets and matched against the observed media streams. So, at least the information about either the session description or the media stream has to be exported to another device before they can be matched.

The next problem is that the device observing the media streams has no clue at which other device the corresponding SIP packets might get observed, to which it could send the stream information. The SIP observing device has at least the session descriptions, but since these usually just include information about the destination of the media streams, this is probably still not enough to tell for sure, where the media stream will be observed. This is even true if knowledge about the network topology is presumed, what again is hardly feasible for a monitoring device.

Similar problems arise for other measurements, for example the *one-way-delay* of media streams from one measuring point to another. For its calculation time-stamps of certain selected media packets are recorded and need to be compared. As these measurements are performed at different devices, they have to be sent to a third device where the actual delay between the two measurement points can be calculated.

What hence is necessary is a monitoring architecture, which supports

- a distributed traffic observation,
- the export of the observed information in regular intervals, and
- an efficient convergence of mutually depending data.

It has to be guaranteed that even without knowledge by the involved devices about the topology the corresponding data will eventually and as early as possible converge at a single device that performs the matching or calculations on this data derived from different probes.

5.3 Solution Building Blocks

Nowadays, general network monitoring is usually done by observing the traffic as flows. Flows consist of packets sharing the same properties. While these properties are typically source, destination and protocol type, they can be any set of packet properties in general. For each observed flow certain counters are maintained, like the number of packets, bytes or transport layer sessions. These counters are stored in regular intervals with the according flow property values. As flows are a widespread concept, there are many advantages in building a solution based on a flow monitoring architecture which supports the requirements of section 5.2. This way it will be easier to make use of synergies with existing general purpose monitoring infrastructures. It will reduce efforts and costs for implementations as well as deployments, since existing code and existing infrastructures can be used, and only the necessary application specific extensions have to be implemented and installed.

5.3.1 IPFIX

IPFIX (IP Flow Information eXchange) is a new set of IETF standards, mainly a protocol [RFC 5101] and an information model [RFC 5102], that defines the transport and storage of general IP flow information. IPFIX offers a distributed, efficient and extensible monitoring architecture.

The IPFIX protocol is based on the latest version of Cisco's NetFlow protocol (v9). NetFlow until now is the most common protocol for flow information exchange. It can be expected that IPFIX will soon be supported by most flow monitoring hardware and software, since due to the similarities the cost of adaptation is quite small, and thereupon replace NetFlow. This makes IPFIX a very promising base for our SIP monitoring scheme, since it has a distributed architecture and will already be deployed in many networks independently.

In IPFIX, the traffic observation is handled by *IPFIX Devices*, which comprise *Metering Processes* that obtain the flow information from direct network observation, and an *Exporting Process* that prepares, schedules and manages the export of this data to one or more receivers as *Flow Records*. They contain the observed flow information for a certain period of time, that is the values of the common packet properties, called *Flow Keys*, and the counters. The receiver of the Flow Records is called *Collector*, which is responsible for centrally processing and storing the flow information. (Fig. 5.1)

IPFIX supports flexible flow definitions while still having an efficient data transport. This is achieved by using a compact data structure in Flow Records which consists of lists of binary encoded values without any separators. In order to be able to decode these structures, they have to be defined by *Template Records*, which describe the order, type and size of each field for certain types of Flow Records. They have to be sent to the Collector before the corresponding Flow Records. The type of a field is given by *Information Elements (IE)*, that are descriptions of the possible fields in Flow Records, like for example `sourceIPv4Address`, `destinationIPv4Address`, `protocolIdentifier` or `tcpSourcePort`. A simple template can look like: `<sourceIPv4Address, destinationIPv4Address, octetDeltaCount>`

Besides a base set, that is defined in the information model, IPFIX supports the definition of new IEs. This is an important feature we will make use of in our scheme, since it enables us to define the IEs necessary for the transport of SIP specific or related information.

5.3.2 Mediators

The original architecture of IPFIX comprises IPFIX Devices including Exporters that send out the flow information, and Collectors that directly receive it. Assuming that one Collector is assigned for a certain network segment to which all the Exporters send their data, this architecture obviously does not scale with a network that grows, either by size or by bandwidth. For large networks this concept is not feasible, because the Collector itself or its network link cannot handle the load anymore. This would result in either being forced to reduce the amount of information that is exported by the IPFIX devices or installing more Collectors that are assigned to segments of the network, which spoils the advantage of having a central data collection.

In order to cope with this problem, the concept of a *Mediator* is currently being developed by the IPFIX working group in several document drafts [Kobayashi et al. 2008a, b]. A Mediator is basically a device which incorporates both a Collecting Process and an Exporting Process together with optional Intermediate Processes. It receives Flow Records from IPFIX devices or other Mediators and can process the data in a number of different ways, such as data reduction by aggregation or filtering, data correlation and combination from different devices, data modification (anonymization for instance), and data storage in distributed repositories. The processed data is exported to a Collector or another Mediator. (Fig. 5.1)

So, Mediators implement a trade-off, distributing the processing and storage needs of fine grained data, while the Collector still receives the amount of data it can handle.

As we will show in the use case examples in section 5.8, Mediators play a particularly important role in our scheme. In many cases data is observed at different probes and has to be correlated in some way. This can be done by Mediators that avoid that all these correlations have to be done in a single central Collector. Since most of the data that has to be correlated is expected to be observed topologically close to each other, distributed Mediators can process the data close to the observation point and thereby distribute most of the workload as much as possible.

Also, in some cases very fine grained and timely data is necessary in order to calculate secondary values and/or metrics. This can produce enormous rates of Flow Records that will easily overload a Collector for larger networks. Mediators preprocess that data, export only selected events and information to the Collector and optionally store the fine grained data in a distributed way in local databases for later access.

Although Mediators increase the complexity of the monitoring management and are additional points of failure, they make it feasible to process all the necessary data of the different use cases in a scalable and efficient way. Therefore the number of Mediators should be in a balance between minimal complexity and the necessary processing distribution.

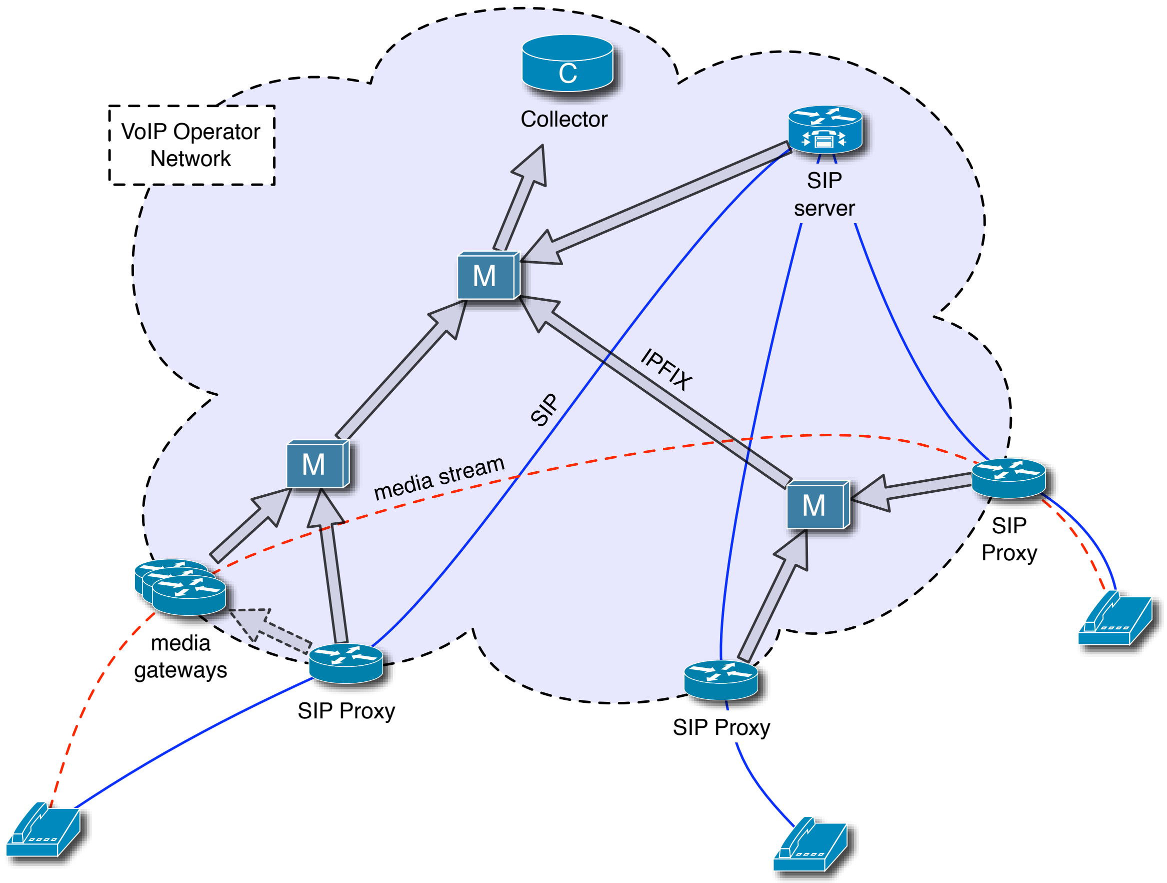


Figure 5.1: Showcase scenario for SIPFIX.

5.4 New Information Element Definitions

In this section we will extend the set of Information Elements of IPFIX, in order to be able to send SIP and media stream related information to Mediators and Collectors. IPFIX supports this either by defining enterprise-specific IEs or by registering new IEs at the IANA registry [RFC 5102]. The new IPFIX IEs we will introduce are either mandatory for the operation of SIPFIX or optional and exemplary, showcasing the possible functionality and feature extensions.

5.4.1 SIP

The following IEs contain information gathered from the header of SIP packets, that is either the first line, called request-line for SIP requests and status-line for SIP responses, or any of the SIP header fields.

`sipFrom`: a string of variable length that contains the value of the *From:* header field of a SIP packet. It consists of the caller's SIP URI and optionally its display name.

`sipTo`: a string of variable length that contains the value of the *To:* header field of a SIP packet. It consists of the callee's SIP URI and optionally its display name.

`sipCallId`: a string of variable length that contains the value of the *CallID:* header field of a SIP packet. It is an arbitrary unique identifier.

These three IEs are mandatory and essential for the architecture, as they are used in data definitions in section 5.5. Together, as a triple, they uniquely identify the dialogs of SIP sessions and are frequently used as unique identifier throughout this scheme. Therefore, we will refer to the IE set `<sipFrom, sipTo, sipCallId>` also as `sipDialogId`, although it is not a real IE but simplifies descriptions.

The following exemplary IEs might be necessary to realize certain functionalities but are not essential for the operation of the basic framework of SIPFIX. Therefore the following list should be seen as a showcase selection:

`sipRequestMethod`: a string of variable length that contains the the first element of the request-line of a SIP request packet. This is the *method* or *type* of the SIP request, like for example »INVITE«, »REGISTER«, »CANCEL«, »BYE« or »ACK«. Of course it can also contain request types of the various SIP extensions like »SUBSCRIBE« and »NOTIFY« [RFC 3265].

`sipRequestURI`: a string of variable length that contains the second element of the request-line of a SIP request packet. This is the request-URI like for example »`sip:bob@example.com`«.

`sipResponseStatus:` an integer that contains the second element of the status-line of a SIP response. This is the numerical status code like »200« for OK or »400« for a bad request.

`sipVersion:` a string of variable length that contains either the third element of a request-line or the first element of a status-line of a SIP request or response, respectively. This is the SIP version, like »SIP/2.0«.

5.4.2 Media

The IEs in this section describe specific properties of media streams related to a SIP session. This information is either gathered from media descriptions in the content of SIP packets, which is usually formatted with the *Session Description Protocol (SDP)*, or from directly observed media packets.

`sipMediaId:` an integer that uniquely identifies a media stream description of a SIP dialog. The media IDs should be assigned in the same order as the corresponding media descriptions in the session description. But nevertheless they are only guaranteed to be constant for a certain media stream description in the scope of a single Exporting Process. A second metering device monitoring the same SIP packets could possibly give the same media stream description a different media ID. One of the purposes of this Information Element is labeling special pseudo flows called *Media Flow Descriptors*, defined in section 5.5.3, which is why this is a mandatory Information Element for this scheme.

`sipMediaProtocol:` a string of variable length that contains the transport protocol from a media stream description. Usually this is extracted from the media descriptions in the SDP data. A typical value is »RTP/AVP«.

`sipMediaType:` a string of variable length that contains the media type from a media stream description. Usually this is extracted from the media descriptions in the SDP data. Typical values are »audio« and »video«.

`sipMediaEncoding`: a string of variable length that contains the encoding name from a media stream description. Usually this is extracted from the media descriptions in the SDP data. Typical values are »GSM«, »PCMU«, »G722« or »H261«.

`rtpPayloadType`: an integer that contains the RTP payload type number. This is either extracted from media descriptions or from observed RTP packets. If it is in the range from 0 to 95 it may refer to static payload type assignments, like 0 for PCMU or 3 for GSM, but in general it refers to dynamical assignments in media descriptions. This means that payload types from RTP streams are of no use without the information from the corresponding media description.

5.4.3 Performance Metrics

For Quality-of-Service or performance monitoring the knowledge about certain metrics is necessary. For SIP applications, this can be either metrics of the signaling performance of SIP or metrics regarding the performance of the related media streams. There are many possible and useful metrics ([Malas 2007], [ITU-T Y.1530]), but it is out of the scope of this work to map a comprehensive metrics list to IEs. In the following we will just define exemplary media metrics that are important for the showcase applications we will describe in section 5.8.

`mediaPacketLoss`: a floating point value that contains the ratio of lost packets to total packets during the observation period of the corresponding flow record.

`mediaDelay{To,From}Terminal`: an integer that contains the one way delay in milliseconds from a media gateway to the terminal (To) and vice versa (From) for the corresponding flow record. This can be measured passively by media stream observation or actively estimated, with loop-back calls or ping, for example.

`mediaDelayMGW`: an integer that contains the one way delay in milliseconds from the ingress media gateway to the egress media gateway for the corresponding flow record. This is usually not measured directly but calculated based on other measurements.

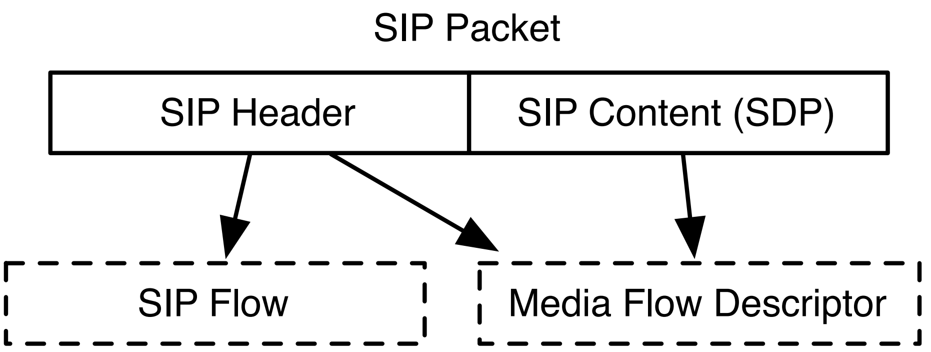


Figure 5.2: Dependencies of SIP Flows and Media Flow Descriptors.

`rtpJitter`: an integer that contains the inter-arrival jitter as defined by the RTP specification [RFC 3550].

5.5 Flow Type Definitions

In order to transmit the information about SIP sessions and their related media streams, our SIP monitoring scheme defines a set of special Flows. These Flows have constraints to make sure that the data can be correctly correlated by a Mediator or Collector afterwards.

5.5.1 SIP Flow

A SIP Flow is a normal Flow of SIP packets, but in addition to the normal fields it must include fields with the Information Elements `<sipFrom, sipTo, sipCallId>` which represent the `sipDialogId`. Therefore the SIP headers have to be parsed. SIP Flow fields may include any number of SIP specific IEs such as those described in section 5.4.1. (Fig. 5.2)

5.5.2 Media Flow

A Media Flow is a normal Flow of media packets. There are no mandatory fields, as these Flows may also be exported by standard IPFIX devices not extended for SIP related monitoring and unaware of the packet content. However, for applications based on media-specific information, like metrics for performance and QoS monitoring, the media probe can gather this information and export it in the Media Flow with media-specific IEs such as from section 5.4.3.

5.5.3 Media Flow Descriptor

The media streams of SIP sessions are defined by media descriptions in the content of SIP packets. This data cannot be exported as normal fields of SIP Flows, as there can be an arbitrary number of media streams described in one single SIP packet. Therefore we define the Media Flow Descriptor, which is not a real Flow based on measured packet properties, but a pseudo Flow that describes an *expected* Media Flow based on media descriptions contained in SIP packets, which is usually an SDP description. (Fig. 5.2) It must include the `sipDialogId` IEs as a reference to the corresponding SIP dialog as well as the `sipMediaId` as a reference to the corresponding media stream.

A Media Flow Descriptor must be specified by an Option Template Record and the `sipMediaId` must be included as a scope field. As it is not a measured Flow it must not contain any kind of counter fields like number of packets or bytes. The `sipMediaId` as scope identifies the Flow as a Media Flow Descriptor and is not allowed as scope of other Flows. If there are several possible variations of an expected Media Flow – for example a `sipMediaEncoding` field is used and there are a number of possible encodings – all the variations can be exported in separate Flow Records with the same `sipMediaId` value indicating that they are all alternatives of the same Media Flow. Besides standard Information Elements, the additional fields of a Media Flow Descriptor typically are based on media descriptions such as from section 5.4.2.

5.6 Extended Device Functionalities

SIPFIX is completely based on the IPFIX standard, so any IPFIX compatible software or device can receive and process the data in a general way. However, in order to obtain the necessary information for the values of the new defined IEs, or to process the data about the SIP sessions in a specific way – for example correlation of signaling and media Flows, security checks or calculation of Quality-of-Service measures – the IPFIX devices require extended functionalities. This section describes the necessary and possible extensions depending on the type of device and its role in the whole monitoring system.

5.6.1 IPFIX Device

An IPFIX Device can either be a dedicated probe, a forwarding device like a router or proxy or even a terminal device. If it is assigned to export SIP Flows it has to obtain at least the information needed for the `sipDialogId`, that is `sipFrom`, `sipTo` and `sipCallId`. Therefore the IPFIX Device has to be extended by a SIP parser to extract all the fields based on the SIP headers as described in section 5.4.1.

An IPFIX Device that is assigned to export Media Flow Descriptors must additionally be able to obtain the media descriptions from the content of the SIP packets as described in section 5.4.2. Nowadays this most probably means it has to be extended by an SDP parser to extract the according information.

An IPFIX Device that is assigned to export Media Flows has no requirement for extensions in general, as there are no mandatory IEs for Media Flows. Nevertheless, it might be necessary to implement extensions for certain applications. For instance, an RTP parser will be necessary for any IEs that depend on RTP, such as `rtpPayloadType` or `rtpJitter`.

Besides those basic extensions, there are a variety of other possible probe extensions.

For Quality-of-Service monitoring the IPFIX Device can gather performance metrics either from the SIP signaling itself or from the Media Flows, such as the examples from section 5.4.3.

If the related SIP packets are not observable, like at media gateways, and for example RTP performance metrics have to be calculated, it might be necessary to have a component that is able to identify Media Flows just by packet inspection and without any signaling information. This of course is a nontrivial task and can be very expensive in terms of processing power.

As an alternative, if it is not feasible to identify media streams in real-time, there can be a media probe component that is able to directly receive Media Flow Descriptors from an associated SIP probe in order to identify arbitrary streams as media streams. However, this requires some knowledge about the topology by the SIP probe, which has to export the Media Flow Descriptors to the correct media probes. (See left bottom of Fig. 5.1)

5.6.2 Mediators

Mediators have the task to preprocess and/or store the Flow data coming from Exporters before sending them to central Collectors, which decentralizes the processing time and data storage and reduces the used bandwidth at the Collector in order to avoid scalability issues. Besides general processing of the IPFIX data, a Mediator can be extended to do specific processing of SIP related data. Among others this can be:

- Calculation of metrics based on values that derive from different probes. For example, the IE `mediaDelayMGW` is based on time-stamps of a packet observed at different media gateways.
- Correlation of SIP Flows with Media Flows by Media Flow Descriptors. This can be useful for adding the `sipDialogId` to Media Flows or to media-based QoS metrics and for calculating the total counters for SIP Flows and their related Media Flows.

- Selection of information that has to be forwarded to the upper next Mediator in the IPFIX hierarchy. For example, if packet time-stamps are only used for the calculation of `mediaDelayMGW`, they are forwarded until a Mediator or Collector is reached that received the time-stamps from both the ingress and the egress media gateway and therefore is able to calculate the delay.

5.6.3 Collectors

Possible extensions of Collectors are general calculation and correlation of data like in Mediators, as long as they are not specific for the export functionality. Additionally, as the Collector is the end of the IPFIX cascade, it is responsible for the final processing, analyzing and archiving of the received data. This ranges from storing compact call records to a real-time display of ongoing calls.

5.7 Recommended IPFIX Extensions

In the following section we will propose the use of two existing IPFIX extensions that optimize the export of the Flow Types in section 5.5. Although not strictly necessary, they are highly recommended, as they improve efficiency and functionality of SIPFIX.

5.7.1 Bidirectional Flows

RFC 5103 [RFC 5103] defines a method to export associated bidirectional Flows (*Biflows*) in a single Flow Record. Two Flows combine to a Biflow, if all non-directional fields directly match, and all source-related fields match the corresponding destination-related field of the other Flow. The Flows are merged by adding special IEs for counter fields of the »reverse« direction from the destination to the source.

This has several advantages: In most cases it is more efficient to assemble Biflows at the measuring device than in a Collector. Also, Biflows share much information, so exporting them as individual Flows creates a large amount of redundant data. But the most important advantage for SIP monitoring is that exporting as Biflows keeps a directional information which otherwise can get lost. For example, if a SIP Flow has the Flow Keys `<sourceIPv4Address, destinationIPv4Address, sipFrom, sipTo, sipCallId>`, it is impossible to tell if this Flow represents packets sent by the caller or the callee, as the `sipFrom` and `sipTo` fields are the same for both directions. If Biflows are used, the source and destination fields are the same for both directions as well and can be associated with the `sipFrom` and `sipTo` fields by *Direction Assignment*.

That is, by using Biflows the SIP Flows of the requests and the responses can be merged in a way that the normal counter fields will refer to the SIP requests, while reverse-counters will refer to the SIP response packets.

5.7.2 Common Properties

If different Flow Records share some properties, that is certain fields have the same values, normally this data is repeated in each single exported Flow Record. This leads to a large amount of redundant data that is being transmitted and wastes bandwidth. Therefore, [RFC 5473] describes a method to reduce the used bandwidth of IPFIX exports by reducing that redundant data. It introduces the special IE `commonPropertiesID`, which acts as an identifier for an arbitrary set of field values. So in case of many Flow Records sharing several field values, a unique `commonPropertiesID` can be assigned to this common set of values in advance. This ID replaces the respective set of values in all the following Flow Records, which are called *Specific Properties Data Records* in this case. The assignment is accomplished by exporting Flows defined by a *Options Template* with `commonPropertiesID` as scope, called *Common Properties Data Records*.

SIPFIX can extensively make use of this method. Foremost the set of IEs called `sipDialogId` described in section 5.4.1 is often used as an identifier throughout this scheme and is even mandatory for SIP Flows and Media Flow Descriptors. Therefore, it is advisable to define an Options Template `<commonPropertiesID | sipFrom, sipTo, sipCallId>` in order to assign a `commonPropertiesID` to each exported `sipDialogId` in a Common Properties Data Record, which is used in all SIP Flows, Media Flow Descriptors and other Flows that contain the `sipDialogId`. Also, if export of SIP Flows is frequently triggered for timely status updates – for example to keep track of the call state via Information Elements like `sipRequestMethod` and `sipResponseStatus` – this results in a lot of small, or even single-packet Flows, which share all fields that are static per session as common properties. Instead of exporting the complete Flows every time, these »status updates« can be reduced to small exports like `<commonPropertiesID, sipRequestMethod, sipResponseStatus>`, where the `commonPropertiesID` represents all the session-static information.

5.8 Use Case Examples

In the following section we will describe some showcase scenarios as possible applications of SIPFIX. They will show the advantages and flexibility of the modular and decentralized architecture of this monitoring framework in order to solve common problems of distributed traffic measurement.

5.8.1 Separate SIP and Media Flows

As SIP is a signaling protocol for setting up media streams, it is very common that the network path of SIP packets is different from the path of media flows. This is due to the fact that the media stream is often set up peer-to-peer, while the SIP communication usually operates over SIP servers at which the users are registered. Furthermore, network operators usually separate the control plane (SIP) from the user data plane (media) by the use of Session-Border-Controllers (SBC), which act as SIP proxies, and dedicated media gateways, which are responsible for the forwarding of the media streams, making it impossible to monitor the whole SIP session at one observation point.

SIPFIX solves this problem by moving the correlation of SIP and Media Flows out of the probe into a Mediator. The SIP probe analyzes the content of the SIP requests, obtains the information about the possible Media Flows for each session and exports them as Media Flow Descriptors to the first Mediator, while other probes observe Media Flows. As early as both the Media Flow Descriptors and the according Media Flow records are received by the same Mediator, the Media Flows can be matched against the observed SIP Flows. The Mediator can then accomplish the same analysis and processing on that data as if a single probe observes both the SIP and media packets.

In a typical situation, the SIP and media probes can be assumed to be network-topologically close. Therefore it is likely that the first Mediator in the IPFIX-forwarding cascade is already the one that can correlate the media with the SIP session. This is beneficial in terms of scalability, as the processing of the data happens close to the source and therefore is well distributed over the infrastructure. Hence, as the network structure grows, additional Mediators can compensate increasing processing needs.

5.8.2 Asymmetric Routing

In case of asymmetric routing the situation can occur, that one probe only sees the SIP requests, while another sees the related SIP responses. In such a case, for instance, it is not possible for a single probe to track the session state for monitoring applications that depend on it, or to calculate the total traffic amount per session.

With our scheme this is still possible if at least both directions are monitored somewhere in the whole monitoring domain. While the unidirectional SIP Flows are forwarded from Mediator to Mediator towards a central Collector, the related SIP Flows will eventually end up in the same Mediator, which can relate them or merge them into one single Biflow.

If the SIP Flows are exported in Biflows in this scenario, the counters for one direction stay empty, which might make it look unnecessary to use Biflows in this case. But still using Biflows has the advantage that it provides the additional information, if the SIP Flow contains requests or responses.

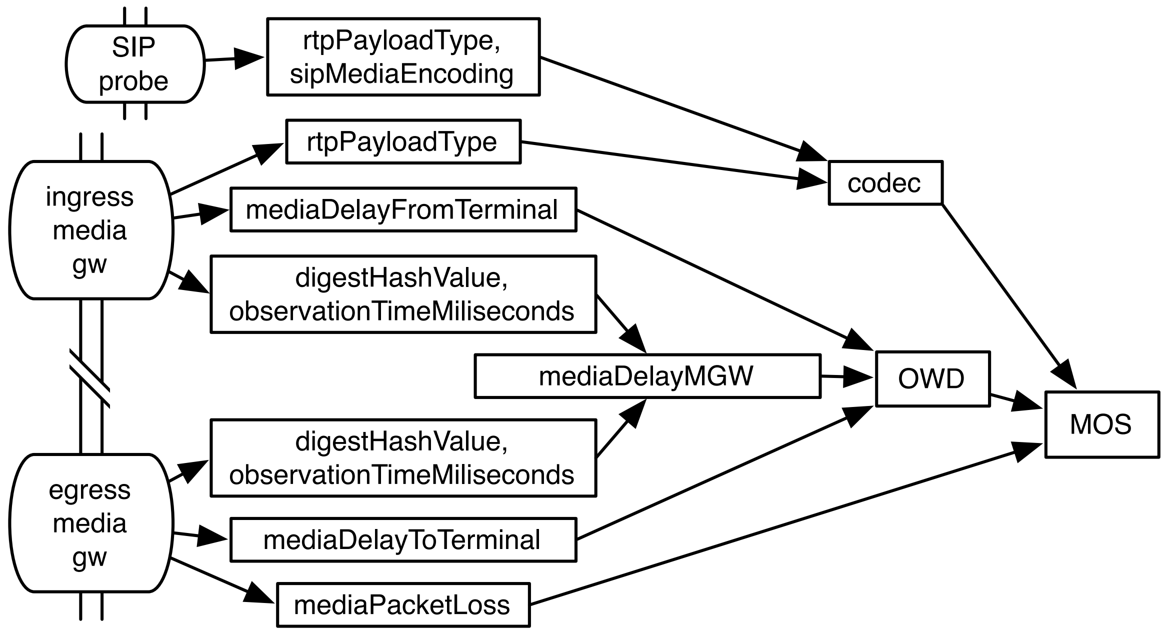


Figure 5.3: Data flow diagram for MOS calculation.

5.8.3 Quality-of-Service Monitoring

QoS monitoring is usually an important issue for VoIP providers and operators, in order to assure the compliance with service-level-agreements or to monitor the performance of the infrastructure for upgrade planning. But in this wide field of measures and metrics, it is difficult to handle it in a general way. Also, the necessary metrics may derive from very different points in the network, which makes it necessary to manage that data remotely in an efficient and scalable way.

SIPFIX supports this approach. New metrics can easily be introduced by the definition of new IEs. And because of the distributed tree-structure of a Mediator-based IPFIX monitoring architecture, the metrics can be collected from any place and still be processed as decentralized and scalable as possible.

As an example, we will outline the assessment of a *Mean-Opinion-Score (MOS)* based on the E-Model[ITU-T G.107], which gives an estimate of the expected call quality of a media stream based on the network parameters *one-way-delay (OWD)* and *packet loss rate* together with the used codec of the voice media stream. We assume the SIP and media traffic is observed separately.

In Fig. 5.3 you can see, how these required values are obtained from the probes.

The media codec is determined by matching the `rtpPayloadType` observed in the media streams against the `rtpPayloadType` and `sipMediaEncoding` pairs in the Media Flow Descriptors exported by the SIP probe. If the `rtpPayloadType` refers to a predefined RTP media profile, in which the codec is defined, the export of `sipMediaEncoding` is not necessary.

The `mediaPacketLoss` is measured and exported by the egress media gateway of a media stream.

The last missing parameter to calculate the MOS estimate – the end-to-end one-way-delay – can be calculated piecewise. We assume that the media gateways can somehow measure the delay to and from the terminal, or at least estimate it with pings or loop-back calls, for example. These values are exported for a Media Flow with the field `mediaDelayFromTerminal` by the ingress, and `mediaDelayToTerminal` by the egress gateway.

In addition, all media gateways deterministically sample certain packets of Media Flows in order to export their values `digestHashValue` and `observation-TimeMiliseconds`. With these values for the same packet from both the ingress and egress media gateways the one-way delay between them can be calculated and exported as `mediaDelayMGW`. In order to make it reusable for other Media Flows that have the same ingress and egress media gateways this could happen with a generic template like `<ingressGWIP, egressGWIP, mediaDe-layMGW>`.

Eventually, with all the values `mediaDelayFromTerminal`, `mediaDelayMGW` and `mediaDelayToTerminal` the overall end-to-end delay of a media stream can be calculated, and together with the `rtpPayloadType`, `sipMediaEncoding` and `mediaPacketLoss` the according MOS value.

All the calculations and correlations from this example can be flexibly distributed over different Mediators in different hierarchy levels.

5.8.4 Security Inspections

Now we will describe some examples how SIPFIX can help apply security constraints or detect anomalies and possible threats.

5.8.4.1 Spoofed Media Sender

Most SIP devices currently do not focus on the security of media-streams. They expect packets to arrive at a certain IP address and port but normally do not inspect the origin. This makes it easy for an attacker to inject media packets in order to take over or disturb the media stream.

If the network is supervised by a SIPFIX infrastructure, this attack would lead to two different Media Flows which both match the same Media Flow Descriptor of a SIP session. This inconsistency can easily be detected by a receiving Mediator, and alarms or counter-measures can be triggered. Even in more sophisticated cases, when the attacker spoofs the IP address of the media source, it still might be possible to detect the fraud. For example, if there are Media Flow Records of the same media stream being received at the same time from different Observation Domains, which topologically cannot be on one path, like three different edge-nodes, for instance, it can be detected as an anomaly and further investigated.

5.8.4.2 Stateful Cross-Protocol IDS

Previous work on VoIP Intrusion Detection Systems proposes analyzing the traffic across different protocol levels and tracking their states [Wu et al. 2004]. SIPFIX fully supports that approach. Mediators can track the states of SIP sessions by the information they receive in SIP Flow Records, and correlate them with information from other protocol levels like TCP. So information about the SIP session and its media streams can provide an additional layer for a cross-protocol IDS architecture.

5.8.4.3 DoS Detection and Prevention

Denial-of-Service detection and prevention has been a hot topic recently. It is generally accepted that it is critical to identify and block the attacks as close to the source as possible in order to not only protect the attacked entity but also the resources of the network infrastructure in between. By the distributed architecture that SIPFIX inherits from IPFIX, it is possible to detect signatures of SIP based DoS attacks close to the origin, in the best case at the entry point of the observed network domain.

Many DoS attacks are *distributed* DoS attacks (DDoS), which means that the origin of the malicious traffic comes from hundreds or thousands of different systems. While the traffic of a single system is negligible, the overall traffic sums up to a harmful amount. To detect that, Mediators and Collectors can sum up the traffic information from all distributed probes by all or selected destinations and trigger alarms and countermeasures as soon as certain thresholds are reached. It is obviously not necessary to have a probe close to the attacked host, which could see the whole amount of attacking traffic, as the detection is based on distributed measurements before the harmful traffic merges. Since the Mediators know at which probe the malicious traffic has been monitored, it can direct the interception as close to the source as possible.

5.8.5 Real-Time Status Display

A very common request of VoIP operators – as for any network operators – is to see »what is going on«, that is to have a real-time sketch of which calls are currently active, in which state they are and how much bandwidth they use, for instance. For such a live status display, it is necessary that some data is forwarded with short delay. On the other hand, other data might not be necessary on a real time basis but only at a later time for archiving purposes. To optimize this, Mediators can select the Flows, which contain IEs that are necessary for the real-time application, and export them on a tight schedule, while other flows are locally stored or exported in longer intervals. For further bandwidth optimization the Mediator can split the

Flows up into time-critical and non-time-critical IEs. This can be done with the help of `commonPropertiesID`, as mentioned in section 5.7.2. Only the time-critical IEs are fast-forwarded in short Flows then, while the rest is retained for aggregation and later export.

5.9 Implementation Challenges

An implementation based on the SIPFIX scheme is still in an early state. So far we focused on the implementation of the efficient packet inspection in IPFIX Devices, which is one of the major challenges for any application layer inspecting network measurement. This resulted into *RTC-Mon* that has been presented by [Fusco et al. 2009] and has been developed in conjunction with SIPFIX in the same research project. It is an extensible framework for developing high-performance real-time monitoring applications. This is partly achieved by executing time-critical code in the kernel space and using specialized device drivers.

However, the gap between network speeds and common cheap memory (DRAM) will further increase. Since sampling is not an option for most SIP monitoring requirements and in order to scale with the increasing amount of data processing this will inevitably lead to more expensive measurement solutions, either by connecting several probes to one observation point or by building high-performance hardware with fast but expensive memory (SRAM).

5.10 Related Work

There are general distributed monitoring solutions like Cisco's NetFlow, which operate on the transport and network layer, and SIP monitoring solutions like Sipient's SIPFlow, which are designed to monitor single SIP servers. However, still there is only few work done trying to integrate those two concepts.

There has been some work on using IPFIX for SIP monitoring. In [Lee et al. 2007] a scheme is presented, which extracts the RTP flow information from SIP packets and proposes IPFIX templates for this information and RTP flows. But since these templates are fixed and are focused on the performance metrics of the RTP flows, the applicability is very limited. Another implementation, which uses IPFIX for the transport of a few RTP QoS metrics by defining new IEs is presented in [Øslebø and Kvittem 2007], but no SIP monitoring is considered here.

In the wider field of VoIP monitoring there is an interesting distributed architecture outlined in [Acharya et al. 2007]. It consists of SIP classifiers located at SIP proxies, which export the gathered information to a SIP Transaction Monitor that tracks the call states. This approach is focusing on SIP itself and does not address the correlation with media stream monitoring.

The solution presented in [Lindh and Roos 2006] is triggering RTP monitoring by information gathered from the SIP signaling, but is depending on direct interaction with the SIP proxy and has not a distributed architecture.

5.11 Chapter Summary

In this chapter we presented SIPFIX, a flexible and distributed scheme for monitoring both the control and user data plane of SIP traffic. It is based on the general purpose monitoring standard IPFIX, which is expected to have soon a high acceptance. This again will make deployment of SIPFIX more feasible and cost-effective. The proposed IPFIX extensions in order to support SIP monitoring comprise

- new Information Elements for SIP and media related data,
- flow types definitions for SIP traffic, media traffic and media descriptions, and
- components which process the new data structures.

The use of Mediators helps to distribute the higher processing and bandwidth needs introduced by the application layer data analysis. In different use case examples we showed that SIPFIX can cope with many of the typical challenges of SIP monitoring, like for example correlation of separated SIP and media flows, end-to-end QoS monitoring and various security inspections.

But still, SIPFIX is just a framework. An important future work is the definition and implementation of application-specific SIPFIX profiles in accordance to the use case examples that maintain interoperability of SIPFIX based components.

Chapter 6

Conclusion

In this chapter we will conclude what we have learned from the work presented in this dissertation, give a concise summary of the original contributions and a short outlook.

In this dissertation we addressed open issues in the field of flow-based network traffic measurements and proposed solutions or improvements for them. (See figure 6.1.)

Although the importance of the Internet and IP networks for the communicative and informational life constantly increases, it is becoming more and more difficult to observe what is happening in the network, because of the increasing bandwidth of the link technologies, the growing network topology and the resulting vast amounts of measurement data. In order to handle these amounts of data, it is important to find better solutions for problems like the analysis and presentation of data in a human-readable format, the long-term storage of the data and the scalability of the measurement infrastructure.

The flow data reduction method Mouse Trapping introduced and evaluated in chapter 3 is based on the observation that only few flows are responsible for most of the traffic. It can – depending on the traffic mix – reduce the flow data by about 90% while only information about 5% of the traffic is lost. Since for most applications a loss is acceptable, this method can improve long-term storage by removing the small flows after some time. It can also improve the scalability of measurement infrastructures if applied in mediators that forward the large flows to the collector.

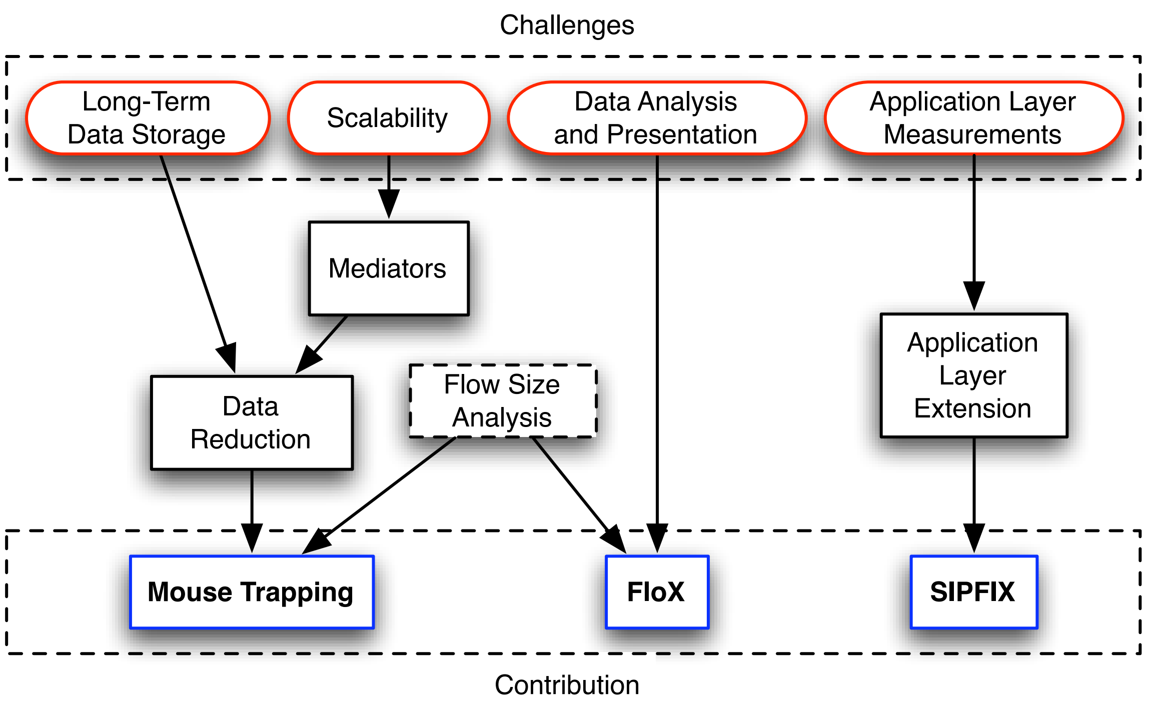


Figure 6.1: Diagram of open issues of flow-based network measurements and their relation to the solutions presented in this work.

The software tool FloX described in chapter 4 solves the analysis and presentation problem for the case of sudden high traffic events by – similar to Mouse Trapping – focusing on the large flows of a dynamically selectable subspace of the flow data in an iterative »drill down« approach.

The wide spread deployment of application layer based server virtualization and overlay network infrastructures make it useful or even necessary to extend the measurement technologies to the observation of application layer parameters. The SIPFIX framework presented in chapter 5 is such an extension for the widely used SIP protocol on basis of the flow export standard IPFIX. Because of its distributed structure, SIPFIX is able to monitor SIP sessions including their related media streams, also for the common case that they take different paths and therefore are measured at different observation points.

6.1 Summary of Contributions

The original contributions presented in this dissertation can be summarized as follows:

Mouse Trapping

1. Evaluation of a reduction method with real traffic measurements that can be highly effective, depending on the traffic mix.
2. Numerical simulations which show that the method works generally for any flow data with power-law distributed flow sizes.
3. Examination of the power-law characteristics of real traffic measurements for various types of traffic.

FloX

1. Design of a concept to examine high traffic events on the basis of large amounts of flow data in an interactive »drill-down« approach.
2. Implementation of FloX, a proof of concept tool that is freely available as a open source web application.

SIPFIX

1. Design of a distributed monitoring architecture for SIP networks that includes the media streams.
2. Extensive description of use case examples for such an architecture.
3. Example of an integrated and distributed measurement infrastructure for both network and application layer.
4. First extensive use of the extensibility of the new IPFIX internet standard.

6.2 Outlook

The specific future work has already been mentioned in each of the chapters summaries. But we will now have some general thoughts about the challenges of future flow-based network traffic measurement solutions.

The general difference between the increase of speed for network links (SRAM) and for ordinary, more affordable memory (DRAM) can be expected to continue. Therefore, without further improvements in the way how measurement data is processed, monitoring solutions will become more expensive in relation to the network infrastructure, since they need much more memory than a simple forwarding network device. Hence, efficient methods to reduce the data to concise digests that contain a maximum of important information, will become even more important than they are today. They have to be efficient in two ways: the reduction ratio must be high and at the same time the »density« of informational value must be high as well.

A problem that the application layer measurement of overlay networks might face is encryption. It is good practice to use end-to-end encryption in terms of data protection and user security, but it prevents the extraction of application layer parameters at the usual observation points of traffic measurement. If peer-to-peer encryption is used, the only possible observation points are at the end hosts themselves, what is usually not feasible. However, in cases where the communication passes central servers and proxies, like normal SIP, for example, these entities are the predestined observation points.

References

Acharya et al. 2007
ACHARYA, Arup; WANG, Xiping; WRIGHT, Charles; BANERJEE, Nilanjan; SENGUPTA, Bikram: Real-time monitoring of SIP infrastructure using message classification. In *MineNet '07: Proceedings of the 3rd annual ACM workshop on Mining network data*, pages 45–50, New York, NY, USA, 2007. ACM. ISBN 978-1-59593-792-6. doi: http://doi.acm.org/10.1145/1269880.1269892. [5.10]

Adamic and Huberman 1999
ADAMIC, Lada; HUBERMAN, Bernardo A.: The nature of markets in the world wide web. Published online, 1999. URL `http://www.hpl.hp.com/research/idl/papers/webmarkets/`. [3.1, 3.3]

Adamic 2000
ADAMIC, Lada A.: Zipf, power-law, pareto - a ranking tutorial. Published online, 2000. URL `http://www.hpl.hp.com/research/idl/papers/ranking/`. [3.2.2]

Aiello et al. 2005
AIELLO, William; GILBERT, Anna; REXROAD, Brian; SEKAR, Vyas: Sparse approximations for high fidelity compression of network traffic data. In *Proceedings of IMC*, pages 253–266. USENIX Association, 2005. URL `http://www.usenix.org/events/imc05/tech/aiello.html`. [3.1.1]

Anderson and Hogrefe 2008
ANDERSON, Sven; HOGREFE, Dieter: Mouse trapping: A flow data reduction method. In *ICIMP '08: Proceedings of the 2008 The Third International Conference on Internet Monitoring and Protection*, pages 17–22, Washington, DC, USA, 2008. IEEE Computer Society. ISBN 978-0-7695-3189-2. doi: http://dx.doi.org/10.1109/ICIMP.2008.23. [3.1]

Anderson et al. 2009
ANDERSON, Sven; NICCOLINI, Saverio; HOGREFE, Dieter: SIPFIX: A scheme for distributed SIP monitoring. In *Proceedings of IFIP/IEEE Symposium on Integrated Management 2009 (IM2009)*, New York, USA, June 2009. (in press). [5.1]

Castro et al. 2004
Castro, Rui; Coates, Mark; Liang, Gang; Nowak, Robert; Yu, Bin: Network tomography: recent developments. *Statistical Science*, 19:499–517, 2004. [2.1.1]

Crovella and Bestavros 1997
Crovella, Mark E.; Bestavros, Azer: Self-similarity in world wide web traffic: evidence and possible causes. *IEEE/ACM Trans. Netw.*, 5(6):835–846, 1997. ISSN 1063-6692. doi: http://dx.doi.org/10.1109/90.650143. [3.1, 3.3]

DAG
The DAG project. URL `http://dag.cs.waikato.ac.nz`. [2.2.1]

Duffield et al. 2001
Duffield, Nick; Lund, Carsten; Thorup, Mikkel: Charging from sampled network usage. In *Proceedings of SIGCOMM Internet Measurement Workshop*, November 2001. [2.2, 3.1]

Duffield et al. 2003
Duffield, Nick; Lund, Carsten; Thorup, Mikkel: Estimating flow distributions from sampled flow statistics. In *SIGCOMM '03: Proceedings of the 2003 conference on Applications, technologies, architectures, and protocols for computer communications*, pages 325–336, New York, NY, USA, 2003. ACM Press. ISBN 1-58113-735-4. doi: http://doi.acm.org/10.1145/863955.863992. [2.2.4]

Estan and Magin 2005
Estan, Cristian; Magin, Garret: Interactive traffic analysis and visualization with wisconsin netpy. In *LISA '05: Proceedings of the 19th conference on Large Installation System Administration Conference*, pages 17–17, Berkeley, CA, USA, 2005. USENIX Association. [4.6]

Estan and Varghese 2003
Estan, Cristian; Varghese, George: New directions in traffic measurement and accounting: Focusing on the elephants, ignoring the mice. *ACM Trans. Comput. Syst.*, 21(3):270–313, 2003. ISSN 0734-2071. doi: http://doi.acm.org/10.1145/859716.859719. [2.2.4, 3.1, 3.3.2]

Faloutsos et al. 1999
Faloutsos, Michalis; Faloutsos, Petros; Faloutsos, Christos: On power-law relationships of the internet topology. In *Proceedings of SIGCOMM*, pages 251–262, New York, NY, USA, 1999. ACM Press. ISBN 1-58113-135-6. doi: http://doi.acm.org/10.1145/316188.316229. [3.1, 3.3]

Fang and Peterson 1999
Fang, W.; Peterson, L.: Inter-as traffic patterns and their implications. In *Proceedings of IEEE GLOBECOM*, volume 3, pages 1859–1868, 1999. URL `http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=832484`. [3.1, 3.3]

Feldmann et al. 2001
Feldmann, Anja; Greenberg, Albert; Lund, Carsten; Reingold, Nick; Rexford, Jennifer; True, Fred: Deriving traffic demands for operational ip networks: methodology and experience. *IEEE/ACM Trans. Netw.*, 9(3):265–280, 2001. ISSN 1063-6692. doi: http://dx.doi.org/10.1109/90.929850. [2.2, 3.1, 3.3]

Flexible NetFlow
Cisco Flexible NetFlow. URL `http://www.cisco.com/web/go/fnf`. [2.3.2]

FloX
FloX (Flow eXplorer): a simple PHP tool to examine large tables of flow data in an SQL database. URL `http://sven.anderson.de/flox/`. [4.1, 4.4]

Fusco et al. 2009
Fusco, Francesco; Huici, Felipe; Deri, Luca; Niccolini, Saverio; Ewald, Thilo: Enabling high-speed and extensible real-time communications monitoring. In *Proceedings of IFIP/IEEE Symposium on Integrated Management 2009 (IM2009)*, New York, USA, June 2009. (in press). [5.9]

GPL
Free Software Foundation, Inc.: GPL: GNU General Public License. URL `http://www.gnu.org/licenses/`. [4.4]

ITU-T G.107
ITU-T: Recommendation G.107: The E-model, a computational model for use in transmission planning, 2006. [5.8.3]

ITU-T Y.1530
ITU-T: Recommendation Y.1530: Call processing performance for voice service in hybrid IP networks, 2004. [5.4.3]

Kobayashi et al. 2008a
Kobayashi, Atsushi; Nishida, Haruhiko; Claise, Benoit: IPFIX Mediation: Framework. Internet-Draft (work in progress) draft-ietf-ipfix-mediators-framework-00, IETF, May 2008. URL `http://tools.ietf.org/html/draft-ietf-ipfix-mediators-framework-00`. [2.3.3.6, 5.3.2]

Kobayashi et al. 2008b
KOBAYASHI, Atsushi; NISHIDA, Haruhiko; SOMMER, Christoph; DRESSLER, Falko; STEPHAN, Emile; CLAISE, Benoit: IPFIX Mediation: Problem Statement. Internet-Draft (work in progress) draft-ietf-ipfix-mediators-problem-statement-00, IETF, May 2008. URL `http://tools.ietf.org/html/draft-ietf-ipfix-mediators-problem-statement-00`. [2.3.3.6, 5.3.2]

Kumar et al. 2005
KUMAR, Abhishek; SUNG, Minho; XU, Jun J.; ZEGURA, Ellen W.: A data streaming algorithm for estimating subpopulation flow size distribution. In *Proceedings of ACM SIGMETRICS*, pages 61–72, New York, NY, USA, 2005. ACM Press. ISBN 1-59593-022-1. doi: http://doi.acm.org/10.1145/1064212.1064221. [3.1.1]

Lee et al. 2007
LEE, Byungjoon; SON, Hyeongu; YOON, Seunghyun; LEE, Youngseok: End-to-end flow monitoring with IPFIX. In *APNOMS*, pages 225–234, 2007. [5.10]

Leland et al. 1995
LELAND, Will E.; WILLINGER, Walter; TAQQU, Murad S.; WILSON, Daniel V.: On the self-similar nature of ethernet traffic. *SIGCOMM Comput. Commun. Rev.*, 25 (1):202–213, 1995. ISSN 0146-4833. doi: http://doi.acm.org/10.1145/205447.205464. [3.1, 3.3]

Lindh and Roos 2006
LINDH, Thomas; ROOS, Emma: Monitoring of SIP-based communication using signalling information for performance measurements. In *ICISP '06: Proceedings of the International Conference on Internet Surveillance and Protection*, Washington, DC, USA, 2006. IEEE Computer Society. ISBN 0-7695-2649-7. doi: http://dx.doi.org/10.1109/ICISP.2006.23. [5.10]

Liu and Huebner 2002
LIU, D.; HUEBNER, F.: Application profiling of ip traffic. In *Proceedings of IEEE LCN*, page 0220, Washington, DC, USA, 2002. IEEE Computer Society. ISBN 0-7695-1591-6. [3.1.1]

Malas 2007
MALAS, D.: SIP End-to-End Performance Metrics. Internet-Draft draft-malas-performance-metrics-08, Internet Engineering Task Force, December 2007. URL `http://www.ietf.org/internet-drafts/draft-malas-performance-metrics-08.txt`. Work in progress. [5.4.3]

Manajan et al. 2001
MANAJAN, Ratul; BELLOVIN, Steven M.; FLOYD, Sally; IOANNIDIS, John; PAXSON, Vern; SHENKER, Scott: Controlling high bandwidth aggregates in the network. Published online, July 2001. URL `http://www.icir.org/pushback/`. [2.2]

NetFlow
Cisco NetFlow. URL `http://www.cisco.com/web/go/netflow`. [2.2.3, 2.3.2]

Øslebø and Kvittem 2007
ØSLEBØ, Arne; KVITTEM, Olav: Extending the IPFIX protocol for better QoS monitoring. In *Proceedings of TNC*, 2007. [5.10]

Patterson and Hennessy 1998
PATTERSON, David A.; HENNESSY, John L.: *Computer organization and design (2nd ed.): the hardware/software interface*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1998. ISBN 1-55860-428-6. [2.2.4]

Plonka 2000
PLONKA, Dave: Flowscan: A network traffic flow reporting and visualization tool. In *LISA '00: Proceedings of the 14th USENIX conference on System administration*, pages 305–318, Berkeley, CA, USA, 2000. USENIX Association. [4.6]

pmacct
pmacct: Promiscuous mode IP Accounting package. URL `http://pmacct.net`. [3.4.1, 4.4]

PSAMP
IETF WORKING GROUP: PSAMP Packet Sampling. URL `http://www.ietf.org/html.charters/psamp-charter.html`. [2.2.4]

RFC 2722
BROWNLEE, N.; MILLS, C.; RUTH, G.: Traffic Flow Measurement: Architecture. RFC 2722 (Informational), October 1999. URL `http://www.ietf.org/rfc/rfc2722.txt`. [2.2.3]

RFC 3265
ROACH, A. B.: Session Initiation Protocol (SIP)-Specific Event Notification. RFC 3265 (Proposed Standard), June 2002. URL `http://www.ietf.org/rfc/rfc3265.txt`. [5.4.1]

RFC 3411
Harrington, D.; Presuhn, R.; Wijnen, B.: An Architecture for Describing Simple Network Management Protocol (SNMP) Management Frameworks. RFC 3411 (Standard), December 2002. URL `http://www.ietf.org/rfc/rfc3411.txt`. Updated by RFC 5343. [2.3.1]

RFC 3550
Schulzrinne, H.; Casner, S.; Frederick, R.; Jacobson, V.: RTP: A Transport Protocol for Real-Time Applications. RFC 3550 (Standard), July 2003. URL `http://www.ietf.org/rfc/rfc3550.txt`. [5.4.3]

RFC 5101
Claise, B.: Specification of the IP Flow Information Export (IPFIX) Protocol for the Exchange of IP Traffic Flow Information. RFC 5101 (Proposed Standard), January 2008. URL `http://www.ietf.org/rfc/rfc5101.txt`. [2.3.3.2, 3.2.1, 5.3.1]

RFC 5102
Quittek, J.; Bryant, S.; Claise, B.; Aitken, P.; Meyer, J.: Information Model for IP Flow Information Export. RFC 5102 (Proposed Standard), January 2008. URL `http://www.ietf.org/rfc/rfc5102.txt`. [2.3.3.2, 5.3.1, 5.4]

RFC 5103
Trammell, B.; Boschi, E.: Bidirectional Flow Export Using IP Flow Information Export (IPFIX). RFC 5103 (Proposed Standard), January 2008. URL `http://www.ietf.org/rfc/rfc5103.txt`. [2.2.3, 2.3.3.4, 5.7.1]

RFC 5473
Boschi, E.; Mark, L.; Claise, B.: Reducing Redundancy in IP Flow Information Export (IPFIX) and Packet Sampling (PSAMP) Reports. RFC 5473 (Informational), March 2009. URL `http://www.ietf.org/rfc/rfc5473.txt`. [2.3.3.5, 5.7.2]

RFC 959
Postel, J.; Reynolds, J.: File Transfer Protocol. RFC 959 (Standard), October 1985. URL `http://www.ietf.org/rfc/rfc959.txt`. [4.5]

Roberts 2000
Roberts, Lawrence G.: Beyond moore's law: Internet growth trends. *IEEE Computer Magazine*, 33(1):117–119, 2000. ISSN 0018-9162. doi: http://dx.doi.org/10.1109/2.963131. [2.2.4]

SerNet
SerNet GmbH, Göttingen, Germany. URL `http://www.SerNet.DE`. [3.4.1]

Shaikh et al. 1999
SHAIKH, Anees; REXFORD, Jennifer; SHIN, Kang G.: Load-sensitive routing of long-lived ip flows. In *SIGCOMM '99: Proceedings of the conference on Applications, technologies, architectures, and protocols for computer communication*, pages 215–226, New York, NY, USA, 1999. ACM. ISBN 1-58113-135-6. doi: http://doi.acm.org/10.1145/316188.316225. [2.2]

tcpdump
tcpdump/libpcap - packet capture project. URL `http://www.tcpdump.org`. [2.2.1]

Wu et al. 2004
WU, Yu-Sung; BAGCHI, Saurabh; GARG, Sachin; SINGH, Navjot; TSAI, Tim: Scidive: A stateful and cross protocol intrusion detection architecture for voice-over-ip environments. In *DSN '04: Proceedings of the 2004 International Conference on Dependable Systems and Networks*, page 433, Washington, DC, USA, 2004. IEEE Computer Society. ISBN 0-7695-2052-9. [5.8.4.2]

Acknowledgments

I want to thank everybody, who supported me and made it possible to finish this work, especially:

- Prof. Dr. Dieter Hogrefe for his support and for accepting me as his PhD student and a member of the Telematics Group,
- Prof. Dr. Bernhard Neumair for being the second reviewer,
- Dr. Johannes Loxen and the SerNet GmbH for funding and a great working environment,
- Dr. Saverio Niccolini and Dr. Jürgen Quittek at the NEC Labs for a great cooperation,
- all my colleagues at SerNet and in the Telematics Group for a great time,
- Dr. Kevin Ivory for his very thorough reviews,
- all my friends and family for just being there,
- and most importantly my wife Chen-Yu for being very patient and taking care of so much.

Akademischer Lebenslauf

Name:	Sven Anderson
Geburtstag:	18. September 1973
Geburtsort:	Freiburg im Breisgau
Staatsangehörigkeit:	deutsch

1984–1993	Besuch des Bismarck-Gymnasiums in Karlsruhe.
1993	Erlangung der Allgemeinen Hochschulreife.
1995–1996	Physikstudium (Diplom) an der Universität Heidelberg.
1996–2003	Physikstudium (Diplom) an der Universität Göttingen.
1999	Erlangung des Vordiploms in Physik.
2003	Erlangung des Hochschulgrades »Diplom-Physiker«.
2003–2005	Wissenschaftlicher Mitarbeiter im *InnenOhrLabor* des Universitätsklinikums Göttingen.
2005–2009	Wissenschaftlicher Mitarbeiter und Doktorand der Telematik Gruppe des Instituts für Informatik, Universität Göttingen.