A Flexible Network Monitoring Tool Based on a Data Stream Management System

Natascha Petry Ligocki, Carmem S. Hara
Departamento de Informática
Universidade Federal do Paraná
Curitiba-PR, Brazil
{ligocki,carmem}@inf.ufpr.br

Christian Lyra
Ponto de Presença
Rede Nacional de Ensino e Pesquisa
Curitiba-PR, Brazil
lyra@pop-pr.rnp.br

Abstract

Network monitoring is a complex task that generally requires the use of different tools for specific purposes. This paper describes a flexible network monitoring tool, called PaQueT, designed to meet a wide range of monitoring needs. The user can define metrics as queries in a process similar to writing queries on a database management system. This approach provides an easy mechanism to adapt the tool as system requirements evolve. PaQueT allows one to monitor values ranging from packet level metrics to those usually provided only by tools based on Netflow or SNMP. PaQueT has been developed as an extension of Borealis Data Stream Management System. The first advantage of our approach is the ability to generate measurements in real time, minimizing the volume of data stored; second, the tool can be easily extended to consider several types of network protocols. We have conducted an experimental study to verify the effectiveness of our approach, and to determine its capacity to process large volumes of data.

1 Introduction

Nowadays, data distribution via computer networks is of paramount importance. It is used not only for commercial purposes, but also for personal use. As a consequence, the quality of a network, including the Internet, is a crucial factor in day-to-day life. Network problems can have a great impact in every type of service, causing problems in several applications, ranging from difficulties in a research lab to complete chaos in stock market and air traffic control. In many applications, the network is expected to be constantly available, and complaints from end users about lack of connection and delays are not uncommon. Another factor that causes apprehension is the vulnerability of the system to network attacks.

Network administrators rely on data provided by monitoring tools [21] to prevent and identify problems such as bottlenecks, bad distribution of resources, misconfiguration, and potential threats to the system security. As an example, the administrator may use TCPDump [20] for testing a firewall or to trace a specific traffic once a problem is identified. In order to characterize traffic flow he may use tools based on Netflow[11], Sflow[18] and SNMP[9]. Typically, in order to accomplish his tasks, the administrator has to master several tools, each of them for a specific purpose. It would be helpful if a single tool were able to generate measurements based on several protocols through an integrated interface. This would facilitate the task of monitoring and minimize the amount of training required from the administrators.

This paper presents a network monitoring tool called *PaQueT* (Packet Query Tool), which has been designed to cover a wide range of monitoring needs. It has been implemented as an extension of Borealis Data Stream Management System (DSMS)[3]. *PaQueT*'s primary goals are flexibility and extensibility. Flexibility is achieved by allowing the user to determine which metrics he/she is interested in by defining a query. This process is similar to writing queries on a database management system, but differs on the type of data processed: instead of considering stored data, it processes streams of packets in a network. The expressive power of the language is similar to SQL, the standard query language for relational databases. As a consequence, the administrator may define queries for obtaining not only data that are frequently provided by network sniffers, but also refined data such as the number of TCP/UDP packets by port in a given time frame, or the top-k source or destination addresses in a local network.

This approach of defining measurements as queries favors reusability. Network administrators often develop scripts for monitoring special scenarios of network traffic. These scripts, usually developed in Perl, are difficult to

reuse. Moreover, maintaining existing scripts is a hard task, since they are not easy to understand and have poor documentation. In *PaQueT*, modifications and improvements can be easily made on the fly by just executing a different query. Another advantage of our approach is the fact that query results are generated in real time, and do not require packets to be locally stored to be processed.

Another key factor of *PaQueT* is extensibility. In its current implementation, *PaQueT* has been designed to monitor a local network, by sniffing packets. Nevertheless, the tool can be easily extended to monitor a backbone, by receiving a flow of Netflow records or SNMP data. Thus, the tool can provide a general framework for monitoring a network, in which the administrator express queries to gather measurements in both local and wide area networks using the same language, and without requiring any local storage.

It is important to notice that some of the advantages the tool provide are consequences of the fact that *PaQueT* has been implemented on top of Borealis. However, a DSMS cannot be considered a network monitoring tool by itself. Thus, the main contribution of this paper is the development of an extension to Borealis DSMS to provide a flexible network monitoring tool. We have conducted an experimental study to verify the effectiveness of our approach, and also to determine its capacity for handling large volumes of data.

The rest of this paper is organized as follows: in Section 2 DSMSs and some related work are presented. The following section describes Borealis DSMS, the extensions required for developing a network monitoring tool, and the architecture of *PaQueT*. Section 4 reports the results of our experimental study. Section 5 concludes the paper presenting some future work.

2 Data Stream Management Systems

Data Stream Management Systems (DSMSs) were designed to provide the same functionality as Database Management Systems (DBMSs), but applied to continuous data flows (*streams*). The main feature of these systems is their ability to provide results in real time, without requiring data to be locally stored. This is particularly useful for network monitoring. However, when the flow rate exceeds the processing capacity of the system, the results may be less precise. Several DSMSs [12, 1, 5, 10, 6, 3] have been proposed in the literature in the last few years.

The main difference between DBMSs and DSMSs is related to how data and queries are handled [16]. DBMSs work with static data and dynamic queries, while DSMSs work with dynamic data and static queries. That is, traditional databases apply different queries on the same set of data; on the other hand, DSMSs apply the same queries on streaming (dynamic) data. A more detailed discussion about stream processing can be found in [22]. Most work on DSMSs are recent, and the majority of these systems are prototypes. Some applications have been developed on top of these prototypes. In [19] a comparative study between the functionality of TelegraphCQ DSMS [10] and T-RAT [24], a tool for network dynamics analysis, is presented. Other study cases include the use of Borealis DSMS in a multi-user game [4], and in a sensor network [2]. Significant results have also been obtained by a commercial DSMS called Gigascope [12]. It has been shown that Gigascope presents several advantages in comparison with other well known tools for network monitoring like Netflow [11].

Most of previous work either implements a specific application, or compares its functionality and performance with other network monitoring tools. In contrast, in this paper we propose an *extension* to Borealis DSMS in order to provide a flexible monitoring tool. Borealis is open-source, and, to the best of our knowledge, it is the only distributed DSMS currently available. To support a distributed architecture, it provides fault tolerance mechanisms, distributed processing, scalability, load balancing, and load shedder facilities [4]. These features are extremely important for improving its performance, especially for data capture, given that targeted applications often have distributed input data sources.

3 PaQueT - A Network Monitoring Tool

In this section we first describe how typical Borealis applications are developed in order to motivate the extensions to the system incorporated by *PaQueT*. Then we describe the tool architecture, and some implementation details.

3.1 Borealis Applications

We will illustrate how an application is developed using Borealis DSMS, considering the following query: "how many UDP and TCP packets passed in a network interface considering time frames of 60 seconds". First, an XML document containing the description of input and output data is defined. Both input and output data consist of a stream of records, where each record is defined as a sequence of elements. This is illustrated in Figure 1. The input schema, called `PacketTuple` (Lines 1 to 29) contains Ethernet and IP headers and an element for each field in TCP and UDP packet headers. In Figure 1 some of the fields were omitted for simplification. For each field it is given a name, a type, and optionally a size. The output schema, denoted as `AggregateTuple` (Lines 30 to 34), contains fields for storing the result of the query: `ip_p` field, which contains the name of the protocol ("tcp" or "udp"), a `timestamp` of the result generation, and the number of packets in the time frame (`numPackets`).

```
1.  <schema name="PacketTuple" >
2.     <field name="timestamp"    type="int" />
3.     <field name="ether_dhost"  type="string"  size="6" />
4.     <field name="ether_shost"  type="string"  size="6" />
5.     <field name="ether_type"   type="string"  size="1" />
6.     <field name="ip_vhl"       type="string"  size="1" />
       ...
12.    <field name="ip_p"         type="string"  size="4" />
       ...
15.    <field name="ip_dest"      type="string"  size="4" />
16.    <field name="tcp_sport"    type="int" />
       ...
24.    <field name="tcp_urp"      type="string"  size="2" />
25.    <field name="udp_sport"    type="int" />
       ...
28.    <field name="udp_sum"      type="int" />
29. </schema>
30. <schema name="AggregateTuple">
31.    <field name="ip_p"         type="string" size="4" />
32.    <field name="timestamp"    type="int" />
33.    <field name="numPackets"   type="int" />
34. </schema>
```

Figure 1. Input and output schema definition.

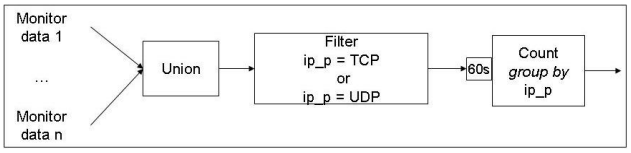

Figure 2. Example of a query diagram.

Next, the user defines a query to obtain the expected result. This can be done using either a graphical interface called *Borealis Graphical User Interface* (Borgui), as illustrated in Figure 2, or directly by an XML file as shown in Figure 3. Some portions of the file were omitted for simplification. In Borgui, queries are defined by boxes, representing operators, and by arrows, representing data flow. Operators supported by Borealis include relational algebra operators such as selection, projection, join, and union, as well as aggregation operators such as count, sum, and average. User defined operators can also be defined. Time frames are defined based either on units of time or on the amount of input data. In the example of Figure 2, a union operator combines the input data of all monitoring points; a filter (or selection) operator drops packets that are not TCP or UDP, and an aggregation operator counts the number of packets of each type sniffed in the last 60 seconds. Given a query in its graphical format, it is translated to a query expressed in XML. The query in Figure 3 expresses the same query of Figure 2, but applied to only one monitoring point.

XML documents containing both input/output schema and the query are given as input to *Marshal*, a tool provided by Borealis which generates code to marshal data between applications and Borealis. In our running example, the application is a C++ program for sniffing packets and breaking them down into fields in the input schema. The marshalling code generated by Borealis are C++ function interfaces to send and receive data from a Borealis engine, both as streams of records, as defined by the input/output schema. After compiling both the C++ application program and the marshaling code, the query can be executed by a Borealis stream processing engine.

```
1.  <borealis>
2.     <input stream="Packet" schema="PacketTuple" />
3.     <output stream="Aggregate"
                 schema="AggregateTuple" />
4.      <query name="NumPacketsUdpTcp" >
5.         <box name="Filter"   type="filter" >
6.            <in stream="Packet" />
7.            <out stream="Filter" />
8.            <parameter name="expression.0"
                    value="ip_p='tcp' || ip_p='udp'" />
9.         </box>
10.        <box name="Counter"   type="aggregate" >
11.           <in stream="Filter" />
12.           <out stream="Aggregate" />
13.           <parameter name="aggregate-function.0"
                    value="count()" />
                 ...
23.        </box>
24.     </query>
25. </borealis>
```

Figure 3. An XML query representation.

There are two major drawbacks for applying Borealis as a network monitoring tool: first, the need for defining both the input and output schema of queries registered in the system; and second, new queries require coding pieces of C++ code and compilation of the program. In this paper, we present *PaQueT*, an extension of Borealis DSMS that allows users to define arbitrary queries on streams of packets with dynamic stream connections. This facility enables *PaQueT* to be used by technicians without programming skills. Moreover, since network protocols clearly define the structure of packet headers, the network administrator can use *PaQueT* to monitor the network in various levels of granularity: from tcp/udp packets to Netflow and Sflow records, using the same query language.

3.2 *PaQueT* Architecture

Figure 4 gives an overview of *PaQueT*. It consists of three main modules: *Borealis* stream processing engine, *IP Tool*, and *Dynamic Stream Interface*. *IP Tool* is responsible for receiving the stream of packets, breaking them down into fields of a predefined *Packet Schema*, and forwarding data to a *Borealis* engine. *Packet Schema* may consist of only fields in TCP and UDP packets, as illustrated in Figure 1, or can be extended to other protocols supported by IPv4 network layer, and other formats such as Netflow, Sflow, and SNMP. Data forwarded from *IP Tool* to *Borealis* consist of a subset of fields in the input schema dynamically determined by *Dynamic Stream Interface* as new queries are registered in the system. Queries are defined using the *Query Register*, which has both a command line interface, and a graphical interface, which is an extension of *Borgui*.

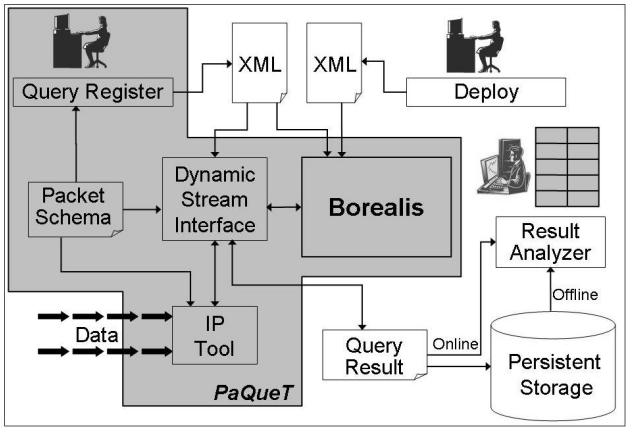

Figure 4. *PaQueT* **architecture.**

In order to register a query in *PaQueT* the user must know the name of the input fields and their meanings to be able to correctly express the expected measurements. *IP Tool* is the module responsible for capturing the input traffic, interpreting packets, and filling up records according to the input schema. It is possible to define more than one input schema for *PaQueT* allowing one to use information from different sources and granularity. For instance, Figure 1 is just a simple example to show *PaQueT*'s capabilities, defining just one schema that covers IP Packets with header from TCP and UDP protocols. Alternatively, it is possible to define two distinct schemas: one for TCP packets, and another for UDP packets. It would also be possible to create schemas describing Netflow records and SNMP data. Our approach is not limited to any particular type of network, but depends only on the available data sources.

Dynamic Stream Interface module receives as input an XML document containing the query definition. It dynamically registers the query in Borealis processing engine, and infers both the input and output schema of the query. The input schema determines the structure of records forwarded from *IP Tool* to Borealis. The output schema consists also of a record of fields, inferred by the *Dynamic Stream Interface*, based on the query definition. Query results can be exported in different portable formats as defined by the user in the query. The output can either be stored in a DBMS or in a file (*Persistent Storage*), or be redirected to a Result Analyzer for generating reports on the fly. Alternatively, the Analyzer may combine incoming data with stored data.

3.3 Implementation

In its current implementation, the *IP Tool* has been developed to sniff packets from a network. It has been implemented in C++, using Pcap [8]. After capturing the packets, their headers are broken down into fields in the input schema. If there are no privacy restrictions, it is also possible to include the contents of the packet to be processed. The information gathered by *IP Tool* is determined by the *Dynamic Stream Interface* based on the registered queries.

Dynamic Stream Interface module has been implemented reusing some code from *Borealis* to infer both input and output schemas. However, it does not depend on the *Borealis* installation. It has been developed such that it is possible to update the DSMS to newer versions without any modifications to *PaQueT*. The module first checks whether all fields used in the query are defined in the input schema. The output schema is determined based on how the input data is used and the operations applied to it. After validating the query and successfully inferring the output schema, the query is registered in the *Borealis* engine.

PaQueT provides a uniform interface for a network administrator to obtain a wide range of monitoring data, which are usually obtained from a number of different tools. The purpose of our approach is to allow the user to obtain customized monitoring information in real time using a simple query language. The experimental study described in the next section shows that *PaQueT* performance for processing large volumes of data is comparable to other monitoring tools that do not provide this flexibility.

4 Experimental Study

A number of experiments have been conducted to validate *PaQueT*. This section presents results from two of them: one to determine the system throughput capacity as the network traffic load increases, and the other to determine how the number of queries registered in the system impact on its CPU usage.

Initial experiments with a simpler version of the tool [17] determined that the results generated by *PaQueT* are accurate, compared with those produced by *Ntop* [14] and *Wire-*

shark [7], two popular network monitoring tools. The primary goal of the experiments in this paper is to determine the applicability of the tool and what is the impact of the tool on an off the shelf equipment.

In all the experiments *Borealis* was executed on an AMD Athlon 3000+ processor with 1 GB of RAM memory, and a 3COM 10/100/1000 NIC; *IP Tool* and *Dynamic Stream Interface* were executed on a 1.5 GHz Pentium 4 with 512 MB of RAM memory and an Intel 100 NIC.

PaQueT **Throughput Capacity.** For this experiment, a query to count the number of TCP and UDP packets per second have been executed in three different scenarios. They differ in the way packets were generated and captured. In the first scenario, denoted as *File*, packets were loaded offline from a dump file; in the second, denoted as *TcpReplay*, traffic was generated by replaying a dump file using TcpReplay [23] at top speed; in the last one, *PaQueT* was executed in a real network using Netcat[15] to burst some TCP traffic between the two computers; we will refer to this scenario as *Real Network*.

Table 1 shows the results in the three settings. The values correspond to the average of 10 executions of each experiment. The first row represents the traffic load measured in mega bits per second (Mbps), and the second row shows the number of packets processed by *PaQueT* in kilo packets per second (kpps). One striking point of these results is the high throughput of *PaQueT* in the *File* scenario, while in the other two the throughput was quite similar. In fact, during the experiments in the *TcpReplay* setting, we noticed that *PaQueT* throughput were almost identical when TcpReplay were set to generate traffic at lower speeds. After analyzing the results, we have concluded that the use of Pcap was the main limitation of our tool. That is, the throughput of *PaQueT* was determined by the number of packets Pcap was able to capture. This can be verified by comparing the second and third rows in Table 1. The fourth row presents the number of packets dropped by Pcap.

Table 1. *PaQueT* **throughput capacity.**

Experiment	File	Tcp Replay	Real Network
Traffic Load (Mbps)	-	47	100
PaQueT (kpps)	45	26	22
Pcap Received (kpps)	-	26	22
Pcap Dropped (kpps)	-	35	12

The Pcap limitation also explains the higher throughput of the *File* scenario since, in this case, Pcap was reading traffic directly from a dump file. As a result, we can conclude that *PaQueT* is able to accurately process all the packets received by Pcap, even at a transmission rate of 45 kpps. In fact, this is close to the rate a single Borealis DSMS engine is able to process. During our experiments, we have increased the number of packets per second a bit higher, and Borealis was not able to process all of them.

It is important to point out that in our experiments *PaQueT* was running on simple computer, without using any distributed features provided by Borealis DSMS, such as the *Load Shedder*. If we consider the default MTU, which specifies the largest packet a network can transmit, *PaQueT* would be able to support speeds up to 520 Mbps. This was above our expectation, given that *Borealis* DSMS is a prototype. Considering that the majority of network protocols does not have such high transmission rates, we can say that *PaQueT* can potentially be applied to any of them.

In [13], it is stressed the importance to shift from thinking in terms of speed to number of packets per second. It is also important to point out that Pcap throughput capacity depends on several issues, including the platform on which it is executed, and on the network interface card and its respective driver. We have tried to modify some parameters in our experiments, but they did not cause any significant changes in the results. Improve Pcap performance or use expensive specialized hardware for traffic capturing are possibilities that are not the subject of this paper. It is important to emphasize that other monitoring tools based on Pcap, such as *TCPDump* [20], *Ntop* [14], and *Wireshark* [7], also present the same limitation, and thus cannot be considered a disadvantage of our approach. Results generated from these tools and *PaQueT* are similar, although they are not directly comparable since they have different approaches for generating the measurements. *Ntop* stores brief information about what is being captured and processes the data when the user requests for updates, generating a number of predefined metrics. *Wireshark* stores all dumped data, which provides the possibility of obtaining any information on later processing. However, storage consumption is very high. In contrast, *PaQueT* can be used to generate arbitrary measurements in real time with low storage cost.

The execution cost of *PaQueT* in all three scenarios described in this section were very low. Memory usage was constant at 5% of the available memory, and in average 15% of CPU was consumed by Borealis DSMS. However, all the experiments were executed with a single simple query. The next experiment describes the effect of increasing the number of registered queries in the system.

The Effect of the Number of Queries. After determining the throughput capacity of *PaQueT*, we now turn to question of how the number of queries being processed by the tool impact on the consumption of the system resources. We have considered queries usually obtained from other network monitoring tools. Although queries differ on their complexity, the results in Figure 5 show that the CPU consumption does not grow linearly with the number of queries. By registering 5 queries, *PaQueT* consumes around 12%

of the CPU capacity; with 10 queries, the consumption is around 22%, and with 20 queries the impact is around 38%. This is because Borealis DSMS applies query optimization techniques on the set of queries, reusing partial results that are common to different queries.

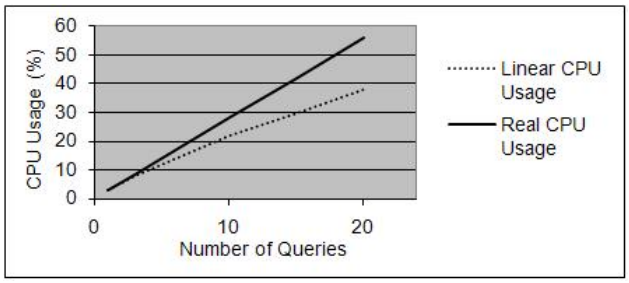

Figure 5. CPU usage X number of queries.

5 Conclusion

We have proposed a flexible network monitoring tool developed as an extension of Borealis DSMS. One advantage of our approach is that it is generic and easily extensible to support a number of protocols. Measurements are defined by constructing queries over a predefined input schema, which describes the structure of packets in the incoming traffic. As a consequence, the network administrator does not have to master several monitoring tools to obtain the desired metrics, but may be able to define them using a single query language. Moreover, data can be analyzed in real time, as opposed to the majority of other monitoring tools.

We have conducted an experimental study to validate our tool. It showed that it is efficient in practice even when the number and complexity of queries registered in the system increase. The experiments have also shown that when *PaQueT* is used as a sniffer, its throughput capacity is limited by the number of packets captured by Pcap. However, this is also the traffic rate supported by other Pcap based monitoring tools, such as *Ntop* and *Wireshark*. Moreover, if *PaQueT* were used for monitoring a wide area network, this upper bound is not a limitation, since Netflow, SFlow, and SNMP traffic is usually lower than the ones monitored by sniffers. This shows that *PaQueT* can effectively be used for monitoring both local and wide area networks.

An important advantage of building *PaQueT* as an extension of Borealis DSMS, is its support for distributed environments. As future work, we intend to extend *PaQueT* to take advantage of Borealis' distributed features, such as load balancing and fault tolerance. Other future work include: implementation of a Result Analyzer, so that the generation of network flow reports is integrated with the rest of the system; and content monitoring, to enable the tool to be used as a firewall.

References

[1] D. J. Abadi et al. Aurora: A data stream management system. In *Proceedings of SIGMOD'03*, pages 666–666, 2003.

[2] D. J. Abadi et al. An integration framework for sensor networks and data stream management systems. In *Proceedings of VLDB'04*, pages 1361–1364, 2004.

[3] D. J. Abadi et al. The design of the borealis stream processing engine. In *Proceedings of CIDR'05*, pages 277–289, 2005.

[4] Y. Ahmad et al. Distributed operation in the borealis stream processing engine. In *Proceedings of SIGMOD'05*, pages 882–884, 2005.

[5] A. Arasu et al. Stream: The stanford data stream management system. *IEEE Data Engineering Bulletin*, 26(1):19–26, 2003.

[6] M. Balazinska et al. Load management and high availability in the medusa distributed stream processing system. In *Proceedings of SIGMOD'04*, pages 929–930, 2004.

[7] Cace Technologies. *Wireshark*, 2007.

[8] T. Carstens. Programming with pcap. `www.tcpdump.org/pcap.htm`, 2002.

[9] J. Case et al. RFC 1157 - Simple Network Management Protocol (SNMP), 1990.

[10] S. Chandrasekaran et al. Telegraphcq: Continuous dataflow processing for an uncertain world. In *Proceedings of CIDR'03*, pages 269–280, 2003.

[11] Cisco Systems Inc. *Introduction to Cisco IOS Netflow - A Technical Overview*, 2006.

[12] C. Cranor et al. The gigascope stream database. *IEEE Data Engineering Bulletin*, 26(1):27–32, 2003.

[13] L. Deri. Improving passive packet capture: Beyond device polling. In *Proceedings of SANE'04*, 2004.

[14] L. Deri and S. Suin. Effective traffic measurement using ntop. *IEEE Communications Magazine*, 38(5):138–143, 2000.

[15] The GNU Netcat project. *Netcat*, 2007.

[16] N. Koudas and D. Srivastava. Data stream query processing6. In *Proceedings of VLDB'03*, pages 1149–1149, 2003.

[17] N. P. Ligocki and C. S. Hara. Uma ferramenta de monitoramento de redes usando sistemas gerenciadores de streams de dados (in Portuguese). In *Proceedins of WGRS'07*, 2007.

[18] P. Phaal et al. RFC 3176: InMon Corporation's sFlow: A Method for Monitoring Traffic in Switched and Routed Networks, 2001.

[19] T. Plagemann et al. Using data stream management systems for traffic analysis - a case study. In *Proceedings of PAM'04*, pages 215–226, 2004.

[20] SANS Institute. *IPv6 TCP IP and tcpdump. Pocket reference guide.*, 2007.

[21] Network monitoring tools. `www.slac.stanford.edu/xorg/nmtf/nmtf-tools.html`, 2007.

[22] M. Stonebraker et al. The 8 requirements of real-time stream processing. *SIGMOD Record*, 34(4):42–47, 2005.

[23] TcpReplay. *TcpReplay Online Manual*, 2007.

[24] Y. Zhang, L. Breslau, et al. On the characteristics and origins of internet flow rates. In *Proceedings of ACM SIGCOMM'02*, pages 309 – 322, 2002.