A Data Stream Management System for Network Traffic Management[*]

Shivnath Babu
Stanford University
shivnath@cs.stanford.edu

Lakshminarayanan Subramanian
University of Califomia at Berkeley
lakme@cs.berkeley.edu

Jennifer Widom
Stanford University
widom@cs.stanford.edu

In this short position paper, we will describe the demands of network traffic management applications and we will discuss how a *Data Stream Management System* can provide a general and scalable platform for deploying these applications.

1 Network Traffic Management

The problem of obtaining the best possible network performance in the growing Internet has given rise to the need for efficient network *traffic management* [2]. Broadly, traffic management can be divided into three tasks: (1) *collecting* data, e.g., network topology and utilization data, router configuration data; (2) *processing* the collected data, e.g., to detect problems such as link failure, to determine the best ways to optimize network performance; (3) *deploying* mechanisms for controlling network traffic. We focus on tasks (1) and (2) in the context of a large Internet Service Provider (ISP) such as AT&T or Sprint.

Data collected to enable traffic management applications in an ISP includes: network packet and flow traces; active measurements of packet delay, loss, and throughput; router forwarding tables and configuration data; and *Simple Network Management Protocol* (SNMP) data maintained by various network elements (e.g., routers, switches) [2, 5]. Since networks need to be running all the time, much of this data is collected continuously (although on different time scales) and results in very large and fast-growing databases. For example, packet traces collected in the Sprint IP backbone amount to 600 Gigabytes of data per day [5].

Different kinds of processing must be performed on the collected data to enable sophisticated traffic management applications. The *network topology* is maintained by joining SNMP data and/or configuration data from different network elements [2]. Statistics of *link and router utilization* are maintained by aggregating packet traces [5] or from SNMP data [2]. *Packet losses*, *per-hop* and *end-to-end delays*, and *network throughput* are measured either by joining packet traces collected from multiple points in the network [5], or by using a dedicated system that generates network traffic to measure these parameters online [2]. The recently proposed *model for traffic demands* in an IP backbone network [4] is computed by joining four different data sources—network flow traces, router forwarding tables and configuration data, and SNMP data.

[*]This work was supported by the National Science Foundation under grant IIS-9811947, by NASA Ames under grant NCC2-5278, and by a Microsoft graduate fellowship.

2 Challenges in Data Processing

Due to the sheer volume of data and complex processing, most current network traffic management applications process the collected data offline [4, 5]. The data is typically loaded into a centralized file system or data warehouse (e.g., *Daytona* [2]) and processed using software toolkits (e.g., *CoralReef* [5], *NetScope* [2]). Offline processing is appropriate for applications using decision-support type queries, e.g., capacity planning and provisioning or determining pricing plans [2]. However, many other traffic management applications would benefit from online data processing. For example, tracking changes in network topology and traffic distribution online would enable congestion cause detection, adaptive intra-domain and inter-domain routing policies (e.g., adjusting OSPF weights and BGP policies [4]), resource allocation mechanisms for guaranteed application-level quality of service (QoS), admission-control and traffic-policing, and detecting denial-of-service attacks [3].

For some of these applications (e.g., network topology maintenance, congestion cause detection, traffic demand computation) data can still be collected and processed in a central place, but they would benefit from a system that performs its processing in a continuous fashion, requiring efficient online data processing techniques. Other applications might require distributed online computation. For example, admission-control and traffic-policing require complex processing over incoming packet streams at different routers (or different machines dedicated to data collection and processing). Collecting and processing data in a central place is difficult due to the real-time response requirements of these applications and the additional load that would be placed on the network. One solution is to maintain a table capturing aggregate properties of the incoming packet stream at each router and transfer (only) significant updates to this table to a central place for further processing. It is essential to keep the table as well as the additional processing overhead at each router within reasonable limits. The structure of an ISP fur-

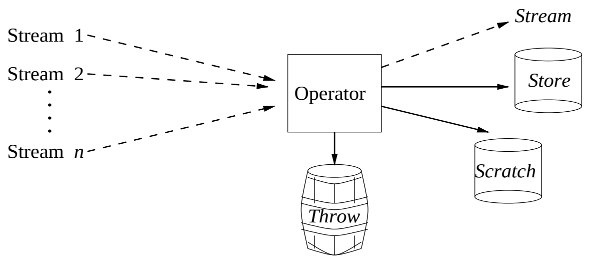

Figure 1: Architecture of a data processing operator in a DSMS.

ther motivates the need for distributed processing. The *backbone routers* of an ISP are significantly more crucial and sensitive than its *access routers* [5]. Therefore, it is preferable to distribute the tasks of data collection and (online) processing to machines monitoring the links between the access and backbone routers [5] or, to a lesser extent, among the access routers, rather than directly among the backbone routers.

3 A Data Stream Management System

To integrate data collection and processing, and to enable online (as well as offline) processing, we propose the use of a *Data Stream Management System* (DSMS) [1] for deploying traffic management applications. The difference between a file system or Database Management System (DBMS) and a DSMS is simple: current file systems or DBMSs expect all data to be managed within some form of persistent *data set*; in a DSMS, the concept of a *data stream*, possibly unbounded, is as important as a conventional stored data set. By nature, a stored data set is appropriate when significant portions of the data are queried again and again, and updates are small and/or relatively infrequent. In contrast, a data stream is appropriate when the data is changing constantly (often exclusively through insertions of new elements), and it is either unnecessary or impractical to operate on large portions of the data multiple times.

The *STREAM* (ST*anford st*RE*am dat*A M*anager*) project at Stanford aims to build a complete DSMS, with functionality and performance similar to that of a traditional DBMS, but which allows some or all of the data being managed to come in the form of unbounded, possibly very rapid, data streams. For details of the envisioned system, see [1]. Data processing operators in a DSMS have the general architecture shown in Figure 1. Each operator works on a set of input data streams and produces an output stream (the *Stream* component in Figure 1). Data may be saved in the *Scratch* component for future use, or it may be discarded (the *Throw* component in Figure 1). Current results are stored in the *Store* component and/or sent to the output stream, which could serve as an input to another operator. These operators are designed particularly for online processing over continuous data streams with bounded amounts of storage. For instance, operators with very rapid or unpredictable input streams could adapt and provide approximate answers in real-time (e.g., by sampling the streams) [1]. In general, the data stream processing model of *one-pass computation* using bounded amounts of storage also provides a scalable alternative for offline data processing over fast-growing databases. Processing traffic management applications over a distributed traffic management infrastructure (recall Section 2) further motivates the use of data stream processing.

Let us revisit some of the network traffic management applications discussed in Sections 1 and 2 supposing we have the support of a DSMS. The collected data is fed to the DSMS as data streams and the applications are enabled using data processing operators based on the general architecture. For example, one operator is a *network topology monitor* operating on a continuous feed of router up/down status and configuration data [4]. Since this operator needs to join data from configuration files of different routers, some data would need to be saved in *Scratch*. *Store* would contain the current network topology, while updates to the topology could be streamed to other operators (e.g., an *intra-domain route monitor* [4]). As a second example, consider traffic demand computation in an IP backbone network [4]. This application can use an *inter-domain route monitor* operator which maintains the set of outbound links carrying traffic from the backbone to specific external IP addresses. The inter-domain route monitor operates on a continuous feed of router forwarding tables (or BGP advertisements) and configuration data [4]. The general operator architecture also captures the needs of applications that maintain aggregate properties of the incoming packet stream at routers (e.g., admission-control and traffic-policing). The DSMS can also be used to integrate and improve offline processing (e.g., decision-support type queries) by applying the same operators to data streams generated from the collected data.

References

[1] S. Babu and J. Widom. Continuous queries over data streams. Technical report, Stanford University Database Group, March 2001. Available at http://dbpubs.stanford.edu/pub/2001-9.

[2] R. Caceres et al. Measurement and analysis of IP network usage and behavior. *IEEE Communications Magazine*, 38(5):144–151, 2000.

[3] N. G. Duffield and M. Grossglauser. Trajectory sampling for direct traffic observation. In *Proc. of the 2000 ACM SIGCOMM*, pages 271–284, September 2000.

[4] A. Feldmann, A. Greenberg, C. Lund, N. Reingold, J. Rexford, and F. True. Deriving traffic demands for operational IP networks: methodology and experience. In *Proc. of the 2000 ACM SIGCOMM*, pages 257–270, September 2000.

[5] C. Fraleigh, S. Moon, C. Diot, B. Lyles, and F. Tobagi. Architecture of a passive monitoring system for backbone IP networks. Technical Report TR00-ATL-101-801, Sprint Advanced Technology Laboratories, October 2000.