The Gigascope Stream Database

Chuck Cranor, Theodore Johnson, Oliver Spatscheck
AT&T Labs – Research
{chuck,johnsont,spatsch}@research.att.com

Vladislav Shkapenyuk
Department of Computer Science, CMU
vshkap@cs.cmu.edu

Abstract

Managing a large scale network requires a network monitoring infrastructure. However, network monitoring is a difficult application. In response to shortcomings in the readily available tools, we have developed Gigascope, a stream database system specialized for network monitoring. In this article, we discuss some of the constraints we faced when developing the Gigascope architecture, Gigascope applications, and how Gigascope is used.

1 Introduction

Modern communications systems are very complex, involving large amounts of expensive equipment running complex protocols, and which must be continuously available. Network monitoring is necessary for many tasks, from finding network bottlenecks to diagnosing network attacks. Researchers at AT&T Labs — Research perform extensive studies using data derived from network monitoring. These experiences showed that existing methods of network monitoring and analysis had serious shortcomings.

One common method of network analysis is to gather data from a network tap using tools such as tcpdump or its underlying library pcap. While this method can provide a great amount of detail for subsequent analysis, there are many problems (as is discussed in [4]). Because so much data is gathered so quickly, the data gathering can be performed only for short periods of time. To extend the sampling period, usually only the network protocol headers are stored, which frustrates analyses which use the application layer headers of the packets (e.g. web or database transactions on a VPN). The data collection is often lossy (i.e., because of buffer overflows) necessitating data munging strategies at analysis time. The resulting data set is collected on a server in thousands to millions of files, and analyzed with hand-crafted programs (for example, Perl scripts). Managing these very large data sets is difficult — data quality problems are common and metadata is quickly lost. As a result, data analyses are often not repeatable. There are some other perhaps less obvious problems. For one, moving these data sets to a central analysis server is very expensive relative to the value of the data. For another, collecting and storing detailed information about network traffic creates privacy and security problems.

A method at the opposite end of the spectrum is to use one of the commercially available network monitoring tools. This approach also presents problems, the most significant of which is the lack of flexibility. Commercially available systems are usually closed, creating data only for their own reporting systems. Making changes to the reports generally requires negotiation with the vendor and winds up being too slow and very expensive (especially for high-speed link monitors). Also, these systems are generally expensive (and AT&T has a *very*

Bulletin of the IEEE Computer Society Technical Committee on Data Engineering

large network) and run on computers provided by the vendor. However these boxes are often not the most appropriate devices for the task.

An intermediate method uses aggregated data sets collected by the router. Two common types of data are SNMP, which collects and distributes a variety of statistics about router usage, and netflow, which is a summary of all flows (packets from a source to a destination) traversing the router. Because these data sets are highly aggregated, they present fewer problems with data management and fewer concerns about privacy and security. However, their coarse nature greatly reduces the range of analyses that can be made. Furthermore, post-collection data analysis is still made difficult by the loss of metadata and the size and unstructured nature of the collected data sets.

To improve our ability to perform various network monitoring tasks, we have developed *Gigascope*, a lightweight stream database specialized for network monitoring applications. Our first deployment of Gigascope occurred in October 2002, and currently (March 2003) we have seven deployments with many more in negotiations. In this article, we describe the application constraints which guided our choice of architecture, a brief discussion of the Gigascope system, and a discussion of Gigascope applications and use.

2 Application Constraints

In our design of the Gigascope architecture, we faced a number of constraints which guided our decisions.

2.1 Data Reduction

The single most critical function of a network monitor is to reduce the amount of data flowing through the system. The earlier the data can be reduced, the less load is placed on the computing system, and therefore the higher the data rate (or the more complex the query set) that can be supported without data loss. As a side benefit, the sooner that irrelevant data can be discarded, the less risk of privacy and security problems. Ideally, the output of the query system is reduced to a small enough size that it can be loaded into a conventional data warehouse, or even viewed directly.

2.2 Flexibility

To a person in the database community, the need for flexibility is self-evident. But in an application-specific domain such as network monitoring, flexibility can be a problem. A more flexible system is generally harder to use, harder to optimize, and easier to abuse than a less flexible system. For example, some systems, e.g. the FLAME architecture [1], propose distributing executable code modules to accomplish network monitoring, but this level of flexibility leads to a system which is very difficult to manage. We have no opportunity to perform critical optimizations and the metadata is lost, leading to problems in the post-collection analysis.

A declarative query language, such as SQL, provides a great deal of flexibility while providing enough structure to enable allow a wide variety of optimizations. However, for network applications SQL is often insufficiently flexible, leading to user frustration and to very complex and inefficient expressions for common network monitoring applications (for example, see the network traffic management queries in [7]). We found that sacrificing the purity of the query language for increased flexibility is sometimes a good tradeoff (as discussed below).

2.3 Query Language

It might be surprising to the readers of a database publication but the choice of query language is an issue. Should it be a standard database language, e.g. SQL, or a special purpose language designed to succinctly and efficiently express network monitoring queries. Consider the examples of Tribeca [8] and Hancock [2], both of which are mostly procedural. In the end we decided that the advantage of using a well-known and well-researched query language outweighed the advantages to be had from a special purpose language.

2.4 Real Time Processing

A stream database system is inherently "push-based" rather than "pull-based", meaning that the system must handle each new packet of data when it is offered, typically by buffering the packet until it can be processed. If the buffer overflows, the system loses data. If the query system is searching for, e.g. multimedia session initiations, losing even a single packet can significantly skew the query results. Therefore the system must incorporate traditionally real-time concerns such as scheduling and buffer sizing.

2.5 Computing Device Selection

Network equipment centers are usually more severe environments than data centers. They are generally small, so that equipment space is at a premium. The network center might be in a remote location and rarely visited by a technician. Even in large network centers, access is generally restricted to qualified technicians. In either case, the equipment must run reliably with only remote maintenance for years. Network centers often have certification requirements for their equipment, which limits the selection of vendors and models. In many network centers, only 48 volt power is available (a telephony standard).

2.6 Reliability

The network monitors are often installed in remote or otherwise difficult to access locations, and must operate for months at a time. One way to ensure reliability is to use reliable server-quality hardware. However, the software system must also be highly reliable, and as a result we generally have a preference for simplicity in our software architecture.

2.7 Efficiency

A critical requirement is efficiency. In part, this requirement is dictated by limitations on devices. Space is usually very limited in network centers: we simply cannot install large computer systems. Cost is another factor. AT&T has a very large network and very many customers. Saving $10,000 on an host computer becomes important when you make hundreds or thousands of installations. Also, it is very convenient to be able to monitor a 100 Mbit/sec network with a laptop. We developed Gigascope to monitor high-speed optical networks, which can carry millions of packets per second at peak usage. Although modern CPUs operate at gigahertz frequencies, there are distressingly few CPU cycles per network bit.

2.8 User Resistance

The target user base (network analysts) are often skeptical of using database technology. In part, they are skeptical because they have not used database tools in the past and are reluctant to change. Also, many attempts to use databases to query network monitor data have failed in the past. A system that is slow, unreliable, or inflexible will be quickly discarded. Instead, we need to show that complex applications can be very quickly developed and that they will be more reliable and have better performance than a hand-crafted system.

3 Gigascope Architecture

We describe the Gigascope architecture more fully in another paper [3], so we give a brief description here and relate it to the application constraints.

Gigascope is a lightweight stream database specialized for network monitoring applications. Gigascope manages streams, or indefinite sequences of tuples. Currently, relations and continuous tables are not supported, and there are no immediate plans to incorporate them (although there are ways to perform certain foreign key joins with tables). Applications receive results by subscribing to the stream which is the output of a query.

Stream database systems must incorporate some notion of time in order to unblock operators such as aggregation and join. One method is to make the dbms a continuous query system, so that all results are continuous tables defined as the result of a query on a recent time window of the input data streams. This approach is well suited to many continuous monitoring queries, but introduces complications for network monitoring (more complex query semantics and less efficient query evaluation). The other common method is to make the dbms a pure stream database – that is, all operators transform input streams into output streams. With a pure-stream dbms, some explicit notion of time is generally needed. Network analysis queries almost always make explicit references to timestamps, so this restriction is not a serious complication. Network data generally contains many types of timestamps and sequence numbers, so Gigascope provides a mechanism for schema annotations to indicate which stream fields behave as timestamps and how.

By pushing timestamp notations into the base stream schemas and using timestamp-ness imputation (similar to type imputation) queries such as

```
Select tb, DestAS, count(*)
From IPV4
Group By time/60 as tb, getlpmid(destIP, 'asn.tbl') as destAS
```

are meaningful, as long as `time` (in this example, a second-granularity timestamp) has been annotated as being a timestamp.

The GSQL query language is in most respects a restriction of SQL. Some of these restrictions are related to programming effort – for example, only two-way joins are supported. Other restrictions are related to the need to find time windows in which to evaluate blocking operators such as aggregation and join. There are also some language extensions, for example the *merge* operator, which is similar to a *union* except that it merges two input streams in a timestamp order.

In order to provide the flexibility required for network analysis, GSQL supports user-written functions and operators. For example, the `getlpmid` function performs *longest prefix matching*, i.e., to find the most specific subnet matching an IP address. Routers use longest prefix matching when doing route lookup, so many network monitoring queries must use this function. The source data is the table `asn.tbl`, which is downloaded from a router. The getlpmid function will read this file when the query starts, build a search data structure, and use this data structure to perform lookups (we have built in the support for this kind of behavior). Network analysts have written highly tuned and very fast C-language modules to do longest prefix matching, one of which is now part of the Gigascope runtime library. The ability to use these special functions is a key part of Gigascope's flexibility. We note that longest prefix matching is a special kind of join, but its expression in relational operators would be complex and inefficient.

Many network analyses require that a network protocol be simulated in part or in whole. Examples include IP defragmentation and multimedia monitoring. To accommodate user-written software, we provide a view mechanism to incorporate user-defined operators. The operator exports a schema, has source queries, and a specification of how selections and projections can be pushed into the operator and its source queries. We have recently added this functionality to Gigascope; and are still experimenting with it.

Gigascope is not a monolithic dbms, instead it operates as a set of cooperating processes. A run-time system evaluates low-level queries on raw packet sources (this important performance optimization is described in [3]). The low level queries generate derived data streams that are consumed by one or more processes, which can be further query processing nodes or application programs. A process which is a query processing node also generates a data stream, and the level of nesting can continue indefinitely deep. A *registry* process contains a mapping from query names to the processes which supply the corresponding stream. To read data from a query, a process (application program or query processing node) finds the data stream source from the registry, then contacts the supplying process to subscribe to the data stream. We used this flexible organization to incorporate user-defined operators.

Gigascope is a compiled query system. When a user submits a set of queries to Gigascope, they are analyzed and optimized, and a set of C and C++ language modules are generated. These are compiled and linked into a run time system, and executed. At this point, the user can access the output data streams. Although the approach of compiled queries places some limits on flexibility, we have found that this approach produces the most efficient system. In a recent application, Gigascope processed 1.2 million packets per second using a moderately priced dual 2.4 Ghz rack mount server.

4 Modes of Use

We have made several Gigascope deployments in the AT&T network and at AT&T customer sites, and we have noticed typical patterns of use. Typically, we define a set of queries, then backhaul the results to a centralized server, where we ingest the data into a data warehouse. We use the data warehouse to generate reports and to populate web pages. Gigascope generally reduces the data volumes to a point where data transfer and data warehouse ingestion costs are quite moderate.

In some occasions, the data volumes are so large, or the backhaul network is so slow, that it is much better to keep the data warehouse on the network monitor. In this case, we just need to spend a little more money on the server's disk drives. The monitor serves web pages, and on demand transfers reports back to an centralized analysis server.

The procedure of gathering Gigascope output and periodically loading it into a data warehouse has worked well for our applications. However there is some delay between data generation and data loading. It would be an interesting experiment to use a continuous query system such as STREAM [6] to monitor Gigascope output.

Another mode of use is to use Gigascope to search for events in a packet stream. We have written a translator to convert Snort [5] rules into GSQL queries, and have written a trigger processing application to interpret the results of the queries. However we have made only preliminary experiments with this type of system.

4.1 Applications

We have used Gigascope in a number of different applications, which present their own challenges.

- **Network analysis:** We have developed a standard query suite to be used for analyzing the health and status of a network. This data is typically backhauled to a data warehouse then correlated with other network data, especially BGP (Border Gateway Protocol) data. The resulting reports give a picture on network health and usage, and correlation with BGP (router) reports often indicates the source of network problems. We have applied this type of analysis to AT&T's network, and to current or potential customer networks.

- **Protocol analysis:** Applications do not just run on a single machine anymore, these days there are many instances of applications which run across large numbers of diverse machines. In these kind of applications, the (wide-area) network is no longer just a dumb pipe, it is a critical part of the properly functioning system. In a recent application of Gigascope, we helped a customer debug performance problems by correlating Gigascope's protocol-level measurements with user experience and with router reports.

- **Research:** The highly flexible nature of Gigascope makes it well-suited as a research tool. Experimental analysis code can be quickly grafted into a high speed monitoring system. Currently we are working with a research team which has developed new algorithms for monitoring the quality of video streams.

- **Intrusion detection:** Network intrusion detection can be accomplished by expressing intrusion rules as GSQL queries and feeding the result streams into a trigger processing application. We have made only preliminary experiments with this kind of application.

5 Conclusions

Gigascope is an applied industrial research project, and as a result has developed in ways different than other recent stream database research projects. The GSQL query language is less expressive and the Gigascope architecture and query processing algorithms are less sophisticated that of, e.g. [6, 9]. However, we have been able to develop a stream database network monitor which operates at very high speeds, is stable enough for unattended operation, and which has been deployed for widely diverse applications. To do so, we have made several query language and architectural innovations, which are more fully described in [3].

References

[1] K. G. Anagnostakis, S. Ioannidis, S. Miltchev, J. Ioannidis, M. B. Greenwald, and J. M. Smith. Efficient packet monitoring for network management. In *Proc. IFIP/IEEE Network Operations and Management Symposium (NOMS)*, 2002.

[2] C. Cortes, K. Fisher, D. Pregibon, A. Rogers, and F. Smith. Hancock: A language for extracting signatures from data streams. In *Proc. Sixth Intl. Conf. on Knowledge Discovery and Data Mining*, pages 9–17, 2000.

[3] C. Cranoe, T. Johnson, V. Shkapenyuk, and O. Spatscheck. Gigascope: A stream database for network applications. In *Proc. ACM SIGMOD Conf.*, 2003.

[4] V. Paxton. Some not-so-pretty admissions about dealing with internet measurements. Invited talk, Workshop on Network-Related Data Management, 2001.

[5] snort.org. Snort home page. http://www.snort.org/.

[6] Stanford Stream Data Manager. Stream home page. http://www-db.stanford.edu/stream/.

[7] Stanford Stream Data Manager. Stream query repository. http://www-db.stanford.edu/stream/sqr/.

[8] M. Sullivan and A. Heybey. Tribeca: A system for managing large databases of network traffic. In *Proc. USENIX Annual Technical Conf.*, 1998.

[9] The Telegraph project. Telegraph home page. http://telegraph.cs.berkeley.edu/.