# Metadata Management of Terabyte Datasets from an IP Backbone Network: Experience and Challenges

Sue B. Moon and Timothy Roscoe
Sprint Advanced Technology Laboratories
1 Adrian Court
Burlingame, CA 94010
{sbmoon,troscoe}@sprintlabs.com

## INTRODUCTION

Network measurements provide insight about real network traffic characteristics which cannot be obtained through modelling or simulation. Frequently, however, subsequent analysis of network data reveals the need for information previously discarded through sampling techniques or insufficient accuracy in measurement. The Sprint IP monitoring project began with a goal of acquiring enough data to answer most questions raised in the course of analysis. This is done by collecting information on every packet without any prior filtering or pre-processing of network traffic.

In this paper we describe the systems issues of analysing data from the Sprint IP Monitoring Project, which collects very large sets of detailed packet-level data from a tier-1 backbone network. We report our early experiences managing these datasets and associated software within a small but growing research group. Based on our experience, we outline a comprehensive framework for efficiently managing the metadata and ultimately the data itself within the project.

## BACKGROUND: The IP Monitoring Project

Sprint operates a tier-1 Internet Backbone network using (at time of writing) Packet-over-SONET (PoS) links of up to OC-48 or OC-192 capacity (2.5 - 10 Gb/s), connecting Points-of-Presence (PoPs) around the United States. Each PoP consists of backbone routers which terminate the backbone links, plus access routers which aggregate low-bandwidth links (OC-3 and under) from customers. Backbone routers and access routers are usually tightly meshed. In addition to backbone and customer links, PoPs generally have links to public and private peering points for connection to other carriers' networks.

Our approach is to collect per-packet header information from multiple links at multiple PoPs simultaneously, and timestamp each packet record using GPS-synchronized clocks[3]. Packet traces are then shipped back to Sprint Labs for off-line analysis.

A passive optical splitter on an OC-3, OC-12 or OC-48 link is connected to a *monitoring system*: a high-end Linux PC equipped with a University of Waikato DAG3 or DAG4 card [2]. The DAG card captures the first 44 bytes of every IP packet on the link, adds 12 bytes of framing information and 8 bytes of timestamp, globally accurate to about 5$\mu$s and synchronized using a GPS receiver in the PoP. The monitoring system transfers these 64-byte packet records over the PCI bus from the DAG card to main memory, then writes them to a RAID array over another PCI bus in 1MB chunks, enabling full-rate traces to be captured to disk even on OC-48 links with minimum-size packets. A *trace* in this context is therefore a vector of 64-byte packet records (on the order of a billion of them). The monitoring system collects packet records up to the capacity of the on-board disks.

While we do not capture packet payloads (simply IP/TCP/UDP headers), no sampling of packets is performed: we record every packet on the link for the duration of the trace run. This results in very large trace datasets: each monitoring system captures between about 50 and 100 gigabytes of data. In addition to the packet traces, we also collect topology information about the PoP configuration and routing tables in effect at the time of the trace. The results of a trace run therefore consist of the following:

- Packet traces from different links in different PoPs, each one between 50 and 100 gigabytes in size.
- PoP configuration information (topology, etc.)
- BGP routing tables downloaded from the routers
- IS-IS contingency tables downloaded from the routers.

Currently, we monitor 9 bidirectional links at one PoP (18 traces at a time). Two other PoPs are in the process of being instrumented, and we will be monitoring about 10 bidirectional links in each. A day-long collection of packet traces amounts to about 1 TB of data, and we expect this to increase to several terabytes per day in the near future.

Analyzing this amount of data poses serious challenges in system design and implementation.

## ANALYZING THE DATA

All data acquired by the monitoring systems is shipped back to Sprint Labs for off-line analysis. The data is stored primarily on two large tape jukeboxes. For processing, traces are loaded off tape onto RAID storage arrays connected to one or more nodes of a 17-machine Linux cluster. We cannot keep all the traces on-line simultaneously.

Analysis involves processing trace files, BGP and IS-IS tables, and other information in assorted ways. Since this is primarily a networking research project, the nature of the analysis is somewhat open-ended. It is not our purpose here to report on the analysis and its results, but we describe some representative operations on traces to give an idea of the systems problems involved in handling them.

*Simple statistics gathering*

Here we process a trace extracting information such as inter-packet delay distribution, protocol types, etc. This is simple sequential processing of a single trace at a time, though in some cases it can generate reasonably large result sets.

*Isolation of TCP flows*

We can process a trace to reassemble each TCP flow. This allows us to infer round-trip time (RTT) distributions for the flows on a link, for instance, as well as generate statistics for TCP goodput. While this is also sequential processing of a single trace, it generates very large result sets and little information is thrown away at this stage, for several reasons (most of the traffic we observe is TCP, for instance).

*Trace correlations*

An important part of our research involves looking at queueing delay distributions through routers. For this we need to take a trace of a link entering a router, and one taken at the same time from a link exiting the same router. From these we generate a list of those packets which both entered on the one link and exited on the other during the period in which both the traces were being taken. Correlating two traces in this way is a frequent operation and is currently performed by building a large hash table of packet records of the first trace, then looking up records in the second trace.

This operation clearly generalizes to the problem of correlating all simultaneous traces into and out of a given router during a trace run, and further to correlating all traces along a packet path in the backbone.

*Generation of network traffic matrices*

A traffic matrix for a network is a two-dimensional array which shows for each combination of ingress and egress PoPs the traffic between them. Traffic matrices are highly dynamic, and may involve multiple internal routes through a network between the same pair of PoPs.

Traffic matrices are extremely useful to know for the purpose of capacity planning and traffic engineering in a network. While we cannot generate precise traffic matrices for Sprint's network without instrumenting all PoPs, by using traces from a small number of PoPs together with BGP tables from the time of the trace, we can infer traffic with a high degree of accuracy.

## EARLY EXPERIENCES

Research work to date on the trace data has tended to proceed in an ad-hoc manner: analysis software has been written from scratch and on demand, storage management (in particular transfering data between disk and tape) is performed manually with few clear conventions on file naming and identification, and individual researchers have tended to produce their own tools and results in isolation. In some ways this has been beneficial: we have generated interesting results quickly, and have developed experience with dealing with the kinds of operations people perform on the data.

However, at the same time we have reached the stage where this approach is becoming unworkable. The kinds of issues that have arisen include:

- The total amount of data we expect to collect over the course of the project is on the order of tens of terabytes. Since this is more than our total on-line storage (currently about 2TB), this raises the problem of when to move datasets from tape to disk, in an environment with multiple users sharing data.
- The results of the early data processing steps are generally needed by most forms of analysis, and often large datasets in themselves. At present there is no facility for sharing result datasets, or even knowing if a desired set of results has already been computed. This results in much lost time and duplicated effort.
- Certain analyses take not only the raw data traces, but also associated BGP tables or topology information as input. It would expedite the analysis process if different types of data can be correlated in a systematic way.
- Given the need to reuse results (due to the cost in time of regenerating them), we need a way of determining which datasets are affected by a bug subsequently discovered in a piece of analysis software. Currently there is none.

As the number of people on the project has grown, these problems have become correspondingly more serious - for small workgroups, informal contacts suffice to keep track of people and data, whereas we expect to have more than 10 people using the data in the near future. We have decided that

the analysis platform needs to be significantly enhanced to support the ongoing sharing of results, software tools, insight and project history among the project members.

## NEW DESIGN

Based on our experience in the early stages of the project, we are currently working on implementing a system for managing the metadata relating to traces and analysis results, and ultimately performing storage management for the project.

Our approach to addressing some of these problems borrows ideas from workflow management and software configuration management systems like Vesta [5]. However, unlike some such systems, the goal here is not to build an environment within which all processing of our trace data occurs: experience shows that such systems are generally heavyweight and ultimately restrictive.

The problem breaks down fairly naturally into three areas:

1. Storage of data (traces, tables, results)
2. Source code maintenance of analysis programs
3. Metadata management.

We discuss only metadata management here. Storage management will continue to be an ad-hoc solution tied to the particular SAN, tape library, etc. in operation and we believe it is best left this way: the datasets are too large for us to manage in a conventional database. Source code management is performed well by systems such as CVS[4], and we do not intend to reinvent the wheel.

Consequently, we concentrate on the management of metadata within the project in this paper. We also emphasize that the following design is in an early stage. We expect the ideas to develop as we gain more experience from building and using the system in the course of the project.

### Metadata Abstractions and Model

We are interested in capturing processes of trace acquisition, analysis tool development, and data processing in ways that are immediately useful to the project while not "getting in the way" of the networking research.

Our proposed system for keeping track of data, results, and analyses within the project is based on four key abstractions of the problem: raw input data sets, analysis programs, result data sets, and analysis operations. We use these to form a dependency graph to represent both the data and results acquired from the project, and the research progress made by the project participants.

*Raw input datasets*

Raw datasets are acquired by the monitoring project rather than generated. This includes the trace files themselves, but also additional data relating to the network, in particular BGP routing tables in effect when the trace was acquired, and information about the topology of both the Sprint backbone as a whole and of individual PoPs being monitored. BGP tables are downloaded in real time directly from routers during a trace collection, while topology information (which changes much less frequently) may be entered by hand.

Each dataset has a varying number of attributes. For example, for trace data we need to know when the trace was taken, its duration, which link on which router the trace was acquired from, what data format the trace is in (the data format has changed as the equipment has evolved).

Input datasets never change. We assume that an input dataset, once acquired, will never be modified.

*Result data sets*

As with raw input data, the results of processing data are also retained as immutable objects (at least, all those results deemed sufficiently important).

The difference between result datasets and raw input datasets is that in principle, given sufficient information about how they were generated, any result dataset can be discarded and subsequently regenerated from the original data.

*Analysis programs*

Each *version* of a program used in the project is represented by an entity in the system. The project has inevitably generated lots of custom software, ranging from scripts for statistical packages like Matlab and S/Plus to C programs for efficiently processing the large out-of-core datasets. All these reside in a version control repository (currently CVS).

The motivation for representing each separate version of a piece of software as metadata is this: the software inevitably evolves as the project progresses, new functionality is added, formats change, and bugs are fixed. Each result obtained in the project is potentially tied to a version of the software that produced it. An extreme case is the discovery of a bug which affects analysis results - it is important to identify which result sets should be considered potentially invalid as a consequence of a programming error.

*Analysis operations*

An analysis operation is a combination of input datasets and programs, which generates a number of result datasets as output. For each operation, we need to keep track of the time, input datasets, specific version of programs, output datasets, and precisely what was done to produce them.

Specifying precisely what an analysis operation consisted of can be quite simple; for example, we might record the Unix command line used to process the data. Given knowledge of the versions of programs used, we can in principle reconstruct any output data from the version control repository and the input data.

*Model*

The four kinds of abstraction above are time-invariant: new elements (datasets, program versions, analyses) are added, but none are ever deleted. They naturally form a dependency graph, where the arcs are analysis operations and the nodes are datasets and program versions.

Given this graph, we can not only keep track of what traces, results and software we have. Other benefits include:

- Much of the data loading and storing is automated that less human intervention is needed and thus expedite the analysis effort.
- Any result can be reliably reproduced from raw data sets.
- Not only the raw data sets and results, but also the procedural information is shared among researchers, adding much efficiency in communication.
- The model provides a sound basis for storage management by providing temporal and spatial constraints between the data sets.

## DESIGN AND IMPLEMENTATION

The dependency graph for the project metadata is easily represented in a relational database schema, and consequently stored in a RDBMS (in our case PostgreSQL, due to ready availability). Not only does this provide a convenient base for building the system, it also allows incremental deployment of functionality when time and resources permit, an important factor in our project.

*Interaction with version control*

The database can interface to the version control repository by referencing modules and major release numbers (for example, using CVS's version tagging capabilities). While it is impractical for the metadata store to track every version of every source file in the repository, it makes sense to track major *releases* of the in-house software[1].

This has the consequence that only analysis operations using "release" versions of software can be tracked by the system. We feel this is a small price to pay for the ability to cache and reproduce results. In any case researchers are free to experiment with new analysis software, they are simply required to check their code in, and thus make it available to others, for their results to be reproducible. This is simply good scientific practice.

*Linkage to data storage system*

Currently trace files are identified by canonical filenames, which encode where and when the trace was taken. We are in the process of making each input dataset self-describing, as with the format used by CAIDA for network traces [1], which will make reliance on a canonical filename less important, and enable us to populate the database of traces semi-automatically.

Having each dataset self-describing facilitates integrity checking. It also makes the metadata independent of file location, an important prerequisite to building a storage management mechanism above the database.

*User interfaces*

There will be at least two user interfaces to the metadata. Using current software engineering tools it is easy to construct sophisticated user interfaces for browsing the metadata, either through a window system or a web browser.

The more challenging part of the interface work is a command-line interface to capture analysis operations. To make the operation as simple as possible to researchers accustomed to using ad-hoc software commands, such as:

```
$ trace_correl -o outputfile trace1 trace2
```

—we would ideally require a minimal change to log the result, such as:

```
$ log trace_correl -o outputfile trace1 trace2
```

The `log` command would ascertain the input datasets, version number of the program being invoked, and the filename of the output dataset. A number of mechanisms immediately suggest themselves: requiring programs to support a `-version` option, having the database maintain a list of MD5 hashes of release binaries, etc. We intend to investigate a number of approaches.

*Incremental deployment and enhancements*

A final, but important, advantage of implementing the metadata store as a database is the ability to deploy new functionality incrementally. This is essential as the monitoring project evolves and we add more functionality to the system. Some of the features that probably will not make it into the first version, but can be added later, include:

- Automatic storage management: have the system make decisions as to when to copy datasets from tape to disk, and when to delete the disk copy. In this way the disk array becomes much more of a cache for the tape archive.
- Result caching: ideally we would automate the caching of results, including fetching previously generated datasets when the system can ascertain that a particular analysis operation would not generate anything new.
- Job scheduling: given the limited resources for processing data (disk space and processors), there is a need for a batch job scheduler which understands dependencies between datasets and the possibility for sharing data between jobs. While this is known to be hard, there is con-

[1] While researchers will write new software and enhance existing programs, we expect that the common case will involve already-built tools.

siderable research literature from the mainframe field in the 1970s on the problem which we could to undertake the task.

- Automation of analysis: operations such as "run program $x$ over every trace file collected on link $y$ in the last 6 months" could be implemented over the metadata store given suitable storage management.

*Caveats*

Perhaps our biggest challenge in designing a system like this is to know where to stop. In the past configuration management and workflow systems have foundered because of a combination of two factors: firstly, they tried to represent too much information about the usage of the system, and secondly, they were not sufficiently flexible in handling events, data and usage patterns outside the scope of the system.

We hope to avoid this trap. The need to deploy the system incrementally "around" our fellow researchers forces a design which does not try to capture all the activities of the research group, while still coping with the real needs of the group.

## CONCLUSION

A number of factors make efficient management of data within the our monitoring project important: the size of raw data sets (on the order of a terabyte each), the changing nature of the network itself over the course of the project, the concurrent development of analysis tools, and the need to be able to reproduce results and reuse them for further analysis.

A further challenge is posed by the requirement that users not be restricted in what they do with the data: this is a networking research project, and to our knowledge analysis of very large, accurately timestamped packet-level traces of an Internet backbone has not been attempted before.

We are attempting to produce an flexible, minimally intrusive system for capturing the complex relationships among datasets and programs for subsequent use. This paper has described the current state of our thinking about this problem.

## ACKNOWLEDGEMENTS

This work has benefitted greatly from discussions with the other members of the Sprint IP Monitoring project: Supratik Bhattacharyya, Imed Chihi, Christophe Diot, Chuck Fraleigh, Gianluca Iannaccone, Ed Kress, Bryan Lyles, Konstantina Papagiannaki, and Nina Taft.

## REFERENCES

[1] CAIDA. Coralreef web page. `http://www.caida.org/tools/measurement/coralreef/`, March 2001.

[2] J. Cleary, S. Donnelly, I. Graham, A. McGregor, and M. Pearson. Design principles for accurate passive measurement. In *Proceedings of the Workshop on Passive and Active Measurements (PAM 2000)*, Hamilton, New Zealand, April 2000.

[3] C. Fraleigh, C. Diot, B. Lyles, S. Moon, D. Papgiannaki, and F. Tobagi. Design and deployment of a passive monitoring infrastructure. In *Proceedings of the Workshop on Passive and Active Measurements*, Amsterdam, Netherlands, April 2001.

[4] P. Lederqvist. *CVS: Concurrent Versions System v. 1.11*, November 2000. Available from `http://www.cvshome.org/docs/manual/cvs.html`.

[5] R. Levin and P. R. McJones. The Vesta Approach to Precise Configuration of Large Software Systems. Research Report 105, Compaq (then Digital) Systems Research Center, 130 Lytton Avenue, Palo Alto, CA 94301, USA, June 1993.