Internet Traffic Warehouse

Chung-Min Chen Munir Cochinwala Claudio Petrone
Marc Pucci Sunil Samtani Patrizia Santa
Telcordia Technologies
Morristown, NJ, USA

Marco Mesiti
Universita' degli Studi di Genova, Italy

Abstract

We report on a network traffic warehousing project at Telcordia. The warehouse supports a variety of applications that require access to Internet traffic data. The applications include Service Level Agreement (SLA), web traffic analysis, network capacity engineering and planning, and billing. We describe the design of the warehouse and the issues encountered in building the warehouse.

1 Introduction

Service and network providers need access to and analysis of network traffic data for efficiency in management of network capacity and engineering, trouble-shooting faults, billing and conformance to service level agreements. Data collection is typically application specific with each application requiring parts of the data in a unique format. For example, a telephony billing application would require call detail records for each toll call [2].

Traditionally, network engineers collect traffic data in flat files as logs or dumps, and use special-purpose tools (such as `awk` and `perl` scripts) and custom programs for analysis [3]. These special purpose tools are generally not re-usable across data sets or for different analyses of the same data because they are customized for specific data formats and for a specific analysis goal.

Even for tools that adopt a DBMS, the data collected are usually "volatile", in the sense that they are thrown away soon after the statistics are computed. These detailed data, however, may contain useful information when gathered over an extended chronological interval. Typically, a catastrophic network backbone failure can be predicted by the presence of small, scattered network problems preceding the big crash.

MOD 2000, Dallas, TX USA

In an attempt to help identify "root causes" of failures and forecast possible network catastrophes, as well as have a single data repository for multiple applications, we have taken the data warehousing approach. The warehouse also paves the way for data mining procedures that are designed to find such prediction rules. The warehouse is built from all the packets at particular collection points in the network. This includes header as well as payload information per packet. The data models allow users to query at the application level (e.g. number of sessions of duration greater than 10 minutes per user) as well as drill down to the packet level.

The rest of the paper is as follows: In Section 2 we give the motivation for using a warehouse; in Section 3 we describe the implementation including data models, cubes and the prototype; in Section 4 we explore two sample applications and in Section 5 we discuss the issues and limitation of the approach. Section 6 contains the status of the project and identifies future directions.

2 Motivation

The level of analysis available by capturing all of the packets used by an application is extremely powerful. Information such as what particular operations have been executed and how long responses took to be generated can be gleaned from the data stream. It can be argued that equivalent information can be generated directly by the end applications if they are sufficiently instrumented and record their behavior in log files. However, not all of the information that may be desired in subsequent analyses may have been anticipated by the instrumentor. For example, very few HTTP servers will include information about aborted downloads, even though this may be very useful in isolating performance bottlenecks or poor web design. Such information may be indirectly obtained from the packet traces. In addition, access to application source code in order to add instrumentation may be unavailable for proprietary reasons. In such cases, the packet trace may be the only

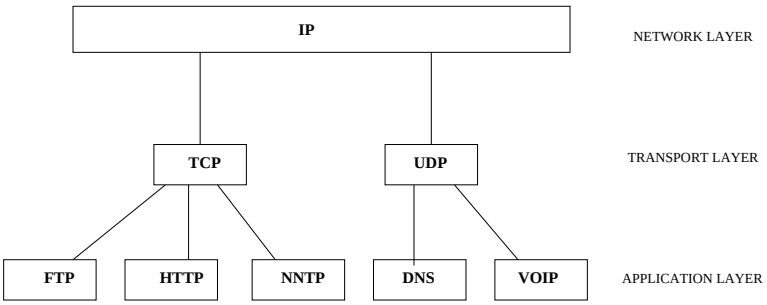

Figure 1: Hierarchical Data Model for Layered Protocols

way to obtain this knowledge. Direct instrumentation may also affect any time sensitivity of an application, or increase its load. Indirect measurement via packet trace analysis is non-intrusive.

Important applications that require access to application level data as well as packet level data are Service Level Agreements (SLA) [1] and Network Trouble Shooting. An SLA is a contract between the service provider and a (usually enterprise) customer on the level of service quality that should be delivered. An SLA may contain several "metrics". Each metric (e.g., available network bandwidth) is associated with a guaranteed performance (e.g., 1.2Mbps), the method of measurement (e.g., 90% of the time on 80% of the user nodes), and a penalty (e.g. $1000) to the provider if the agreement is not met. SLAs give providers a competitive edge for selling customized services into the consumer market and maintaining a high level of customer satisfaction. To track SLAs, the service provider must monitor application-specific performance metrics which generate per user reports on satisfaction/violation of the metrics. In addition, the provider must have the ability to drill down to detailed data in response to customer inquiries.

A warehousing approach to storing data combined with the flexibility of OLAP allows a single data repository and system to support diverse applications. An application can use the traffic warehouse by defining its view on the base tables. A warehouse also provides a "historical" perspective on network traffic and allows construction of data marts. These data marts can be used for determining network growth and hence forecasting of demand. Similarly data mining can be used to correlate faults and/or relation of traffic patterns at different time periods or in the presence of promotions that cause service overloads. Service overloads can also be caused by denial of service attacks.

3 Data Warehouse Implementation

The system requires a traffic data collection mechanism as well as methodology for populating that data into the warehouse. Data models for the different network protocols also need to be defined. In this section, we describe the overall system with some insights into the complexity of the implementation.

3.1 Data Acquisition, Transportation and Integration

The data sources of our Internet traffic data warehouse exhibit some distinctive characteristics. First, the Internet traffic is a dynamic, ever-increasing data stream, as opposed to "static" data sources that are typically stored in relational databases or flat files [4]. We have written a suite of customized programs that read and filter data from the data collection routines into several output streams, each of which corresponds to a specific application such as a single HTTP or FTP operation. The input stream is reconstructed from the low level packet data contained in the trace. Protocol specific filter programs then parse this data and extract the data that has been identified as relevant for the given protocol. For example, an FTP session will record the name, size and transfer rate of the files it passes, while the HTTP filter will record the number and type of web objects contained in each page. The output is flushed to flat files and is further cleansed and correlated before they are imported into the warehouse.

Due to the continuous nature of traffic, we can only keep the data in a staging area for a limited period. The data must be processed (abstracted, cleansed, transferred, and imported into the warehouse) before the storage is re-claimed to hold the next batch of data. We have found no ETL (Extraction, Transformation, and Loading) tools that provide such capabilities.

Preserving the privacy expectations of the users of networks and applications is also of prime concern In particular, the full traces contain system passwords, private correspondence, proprietary documents and other sensitive data. Even the source and destination IP addresses of individual sessions convey information that may be considered private, especially since in today's computing arrangements, an IP address may uniquely identify a user.

While it is possible to completely obscure all of the

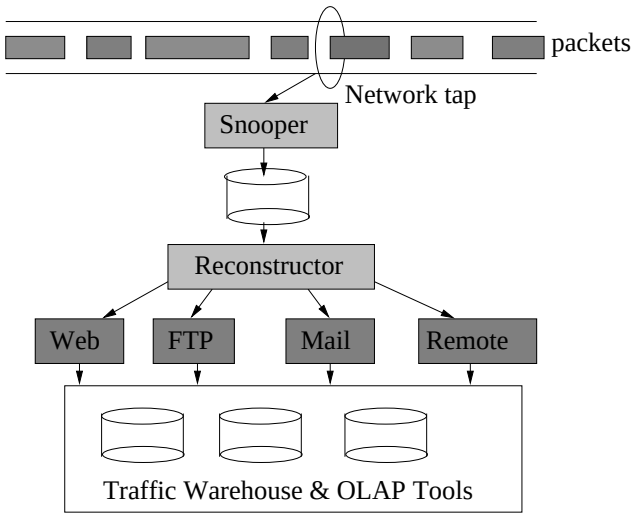

Figure 2: System Architecture

sensitive information in the header and payload of a packet stream, to do so would remove much of the valuable information. A balance must be found between the privacy requirements and analysis needs. While we want to conceal IP addresses so they cannot be used to identify specific sites, we also want to be able to detect multiple accesses to the same site, and therefore must use a consistent mapping scheme from original to obfuscated addresses.

3.2 Data Model

Logically, the warehouse is divided into two levels: the Network Layer (NL) and the Business Layer (BL). The Business Layer model consists of applications that can create their own views on the base relations in the NL model. Rules for materialization are defined under the control of the business application and based on business processes. For instance, billing applications will require views to be materialized at the end of a billing cycle, although they may choose to do it periodically or on demand for pay-as-you-go billing. With this design, new applications can be easily created. This enables service providers to deploy new or customized services based on market need.

The NL defines a data model for multi-layer network protocols (Figure 1). In our instance, the NL contains the network layer (with IP protocol), the transport layer (with TCP/UDP protocols), and the application layer (which models application-specific protocols such as HTTP, FTP, H.323). Each protocol in a layer is made of a set of entities that can be easily added or detached from the main schema. The multi-layer structure allows users to easily navigate and retrieve information at different levels of granularity. In our model, each protocol, such as HTTP, is identified by a port number. Any user-defined application that is similarly identified can be integrated by adding appropriate custom filters to extract header and payload information.

3.3 Data Cubes

To facilitate OLAP queries, we have also created data cubes out of the base relations in the hierarchical network model. We chose to use an ROLAP data mart building tool for this purpose. Figure 3 shows an example data cube concerning Web access statistics. The fact table contains detailed IP records pertaining to HTTP that are selected from the base IP table. The dimension tables `REQUEST_RESPONSE` and `URL` are derived from the suite of HTTP base tables (to be described in Section 4.1).

With this data cube, we may answer questions related to Web page access easily. Consider the question: what is the total size of Web pages under domain www.research.telcordia.com that are accessed and transferred to users (customers) during the last month? This can be answered by slicing on the `CUSTOMER`, `URL`, and `TIME` dimensions, and computing the `SUM` aggregate on `IP_PACKET.packet_length`.

3.4 System Architecture

The system (Figure 2) consists of four phases: collection, re-construction, data loading and analysis. The overall process begins with low level access to the traffic data stream. There are several collection options available here. One technique is to use the tcpdump command available on most Unix systems to extract IP data directly from a computer attached to the network to be monitored. We have also used custom designed hardware that interfaces directly to T1 and OC-3c communications lines and copies traffic directly to tertiary storage. Depending on the data rates and duration

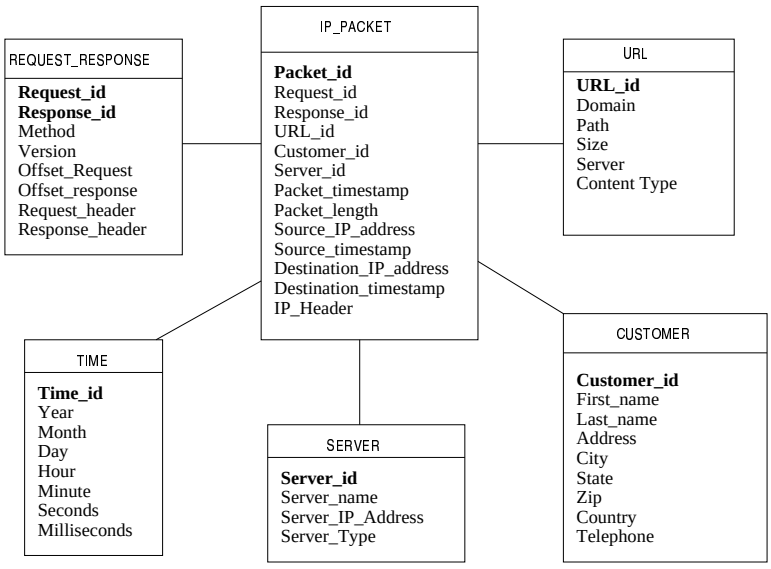

Figure 3: Star Schema for HTTP/IP data

of measurement, the required storage capacity can be enormous; terabyte sized data sets are to be expected.

The collected information represents the data that passes by on the physical communications link, not the data that an application would necessarily see. It also contains the interpersed traffic of hundreds or thousands of individual sessions whose data are interspersed across different applications. These data need to be reconstructed into the form that an application would encounter. This is done using a program that captures all the data embedded in the various protocol semantics in the 7-layer protocol stack. It takes as input a packet, parses the data in the stream (by identifying the port numbers of various application protocols), and generates data pertaining to the protocols (e.g. HTTP, FTP, TCP, IP). The program operates in the following manner: First, individual TCP and UDP sessions are identified in the data stream. Next, a sufficient number of packets are located to perform any necessary packet reassembly, duplicate detection and discard, and out-of-order corrections. Since both sides of a session need to be passed to the application filter, the packets traveling in each direction on the link need to be correlated and made available to the filter in the proper time sequence.

The port number found in the TCP or UDP header is used to select an appropriate application filter that will process the bidirectional traffic. Each filter is specialized to parse and analyze the stream information according to the behavior of the application. For example, the FTP filter will parse and record the commands and responses used in its file transfer protocol, but will skip over the bulk data involved in a file transfer.

The HTTP filter parses the HTTP protocol data, including the use of persistent connections for multiple requests. It also decodes the HTML information contained within a request. It uses a regular expression parser to detect comments, URLs, scripting languages, embedded objects and other needed for the further analysis.

The output is staged into temporary files that are subsequently loaded into the warehouse. If one were loading HTTP data, the data would be parsed and multiplexed into four files: `IP packets`, `TCP connections`, `HTTP requests` and `HTTP responses`. This generates hundreds of MB of data to be loaded into the data warehouse per day. Once the files are generated in a staging area, they are loaded into the data warehouse as defined above.

4 Example Applications

We have actually collected data and analyzed multiple applications. In this section we describe two examples: Web Traffic Analysis and Voice over IP.

4.1 Web Traffic Analysis

We collect the data for Web traffic analysis from two sources: web server logs and an in-house developed network traffic filtering utility program as described above. The web server logs are readily available from most HTTP servers, in which they record some statistics about the HTTP requests. The logs we use come from a number of Apache HTTP servers. The data in web server logs contain information on every incoming HTTP request and the corresponding response, such as users (in terms of IP address or domain name) access patterns to pages (in terms of URL address).

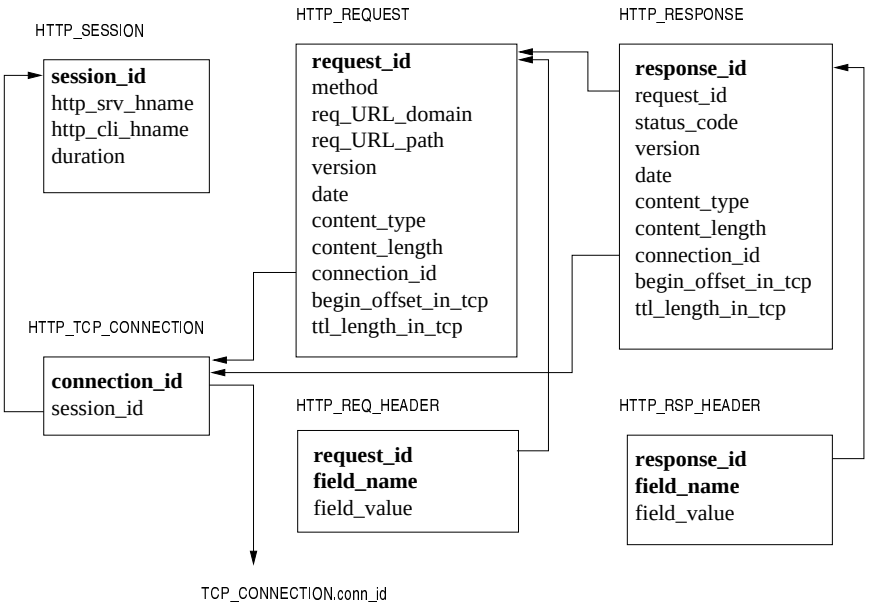

Figure 4: Schema of HTTP traffic data

While providing useful information, data contained in HTTP server logs pertains to the HTTP protocol layer only. This is not enough information for detailed analysis of HTTP traffic. (Recall, HTTP is built on top of TCP and IP protocols). In addition, some header information defined in the HTTP protocol is not captured by the server logs. This includes information such as content type, content length, HTTP version and correlation between the lower layers (TCP/IP) and the application layer (HTTP). This information is of crucial importance to system administrators responsible for system performance monitoring and resource planning.

Figure 4 shows the schema of the HTTP portion of the data model described in Figure 1. Primary keys are identified in boldface, whereas the arrowed lines indicate the foreign key relationships among the tables. All foreign key relationships are many-to-one except the one between `HTTP_REQUEST` and `HTTP_RESPONSE`, which is one-to-one. The `HTTP_TCP_CONNECTION` table identifies individual HTTP connection, which coincides with a TCP connection. Each such connection points to a corresponding TCP connection record in the `TCP_CONNECTION` table (which is not shown here).

An HTTP connection may contain several round trips of requests and responses (the so called "persistent" HTTP connection, as supported by HTTP version 1.1). Each record in the `HTTP_REQUEST` table represents a request from a browser. It is associated to a corresponding response in the `HTTP_RESPONSE` table. Each request/response is also associated with a number of HTTP headers. Finally, we create a table called `HTTP_SESSION` that introduces the fuzzy notion of "HTTP session", which is defined to be a sequence of requests from a visitor that are close in time.

On top of the base tables that host the protocol-wise data, we define star schema that facilitates analyses from different views. The star schema is the one described in Figure 3, and its purpose is to provide the appropriate structure for OLAP operations.

Figures 5, 6 and 7 give the look and feel of the OLAP output. Figure 5 shows the response time of requests made to various web sites. The first column lists the domain names of the web sites (consider this the "`SERVER`" dimension from Figure 3). Listed under each domain name are the individual HTTP connections and requests made to that site (consider these the "`REQUEST_RESPONSE`" dimension from Figure 3). For example, the bottom of the crosstab shows the first five connections (connection ID 912, 925, 926, 927 and 971) to `cnn.com`. Listed under each connection are individual HTTP requests (e.g. connection 971 contains three HTTP requests). The column "Content Length" indicates the total size of the file(s) getting transferred in each request, connection, and by each web site (during the observed period). The next column gives (total) round-trip response time of the transfers (computed from the "`TIME`" dimension of Figure 3). The last column measures the effective bandwidth which is computed by dividing the previous two columns. We note that the user can further drill down to TCP or even IP levels (omitted here) for more detailed analysis.

Figure 6 lists the number of different HTTP links embedded in the web pages accessed by individual browsers. These numbers are summarized at different levels: HTTP connection, HTTP request, content type,

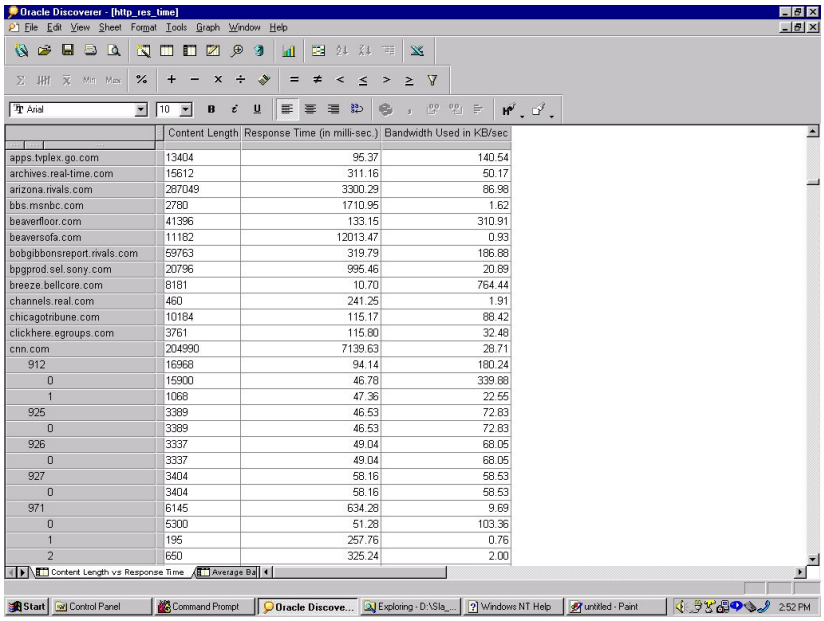

Figure 5: HTTP request response time

link type, and link address. For example, the first group of records indicate that 31 links are embedded in the web pages requested by connection 37. And the response to the fourth request (request 4) in the connection contains 12 embedded images and 7 hyperlinks. The number of links in a web page can be used by providers and caching server providers to determine the type of web page and thus use application specific information to cache.

Finally, Figure 7 gives the number of visits to various Web sites. One can also drill down to find out the visitors of the Web sites. The number strings in the first column under `home.netscape.com` are scrambled IP addresses of the visitors. The IP addresses are scrambled to maintain privacy.

4.2 Voice over IP

Many vendors are now attempting to emulate traditional telephony over the internet. However, to be competitive in the marketplace, the Quality of Service (QoS) of voice over IP should be equal to, if not better than traditional telephony. In traditional telephony (circuit switched network), a dedicated path between the two endpoints (callers) is set up with all required resources. The allocated resources are sufficient to handle continuous voice traffic from either endpoint. If a connection is made the quality of service is guaranteed. The call set-up time has real-time constraints so that either a connection is made in reasonable time (around 300 msecs) or not at all. If resources are not available either a fast busy or a voice recording is heard by the originating caller.

The packet world does not have the same QoS guarantees that a circuit switched world has. Thus, mechanisms for determining QoS need to be introduced. Voice quality is governed by jitter, loss and delay of packets and can be accurately quantified by data collection and monitoring. Call set-up time can be determined either by analyzing the log files generated by telephony "soft-switches" or examining the data traces.

Different paradigms exist today for setting up internet telephony calls. The most common protocols being used are ITU-T H.323, MGCP and SIP. In our study, we modeled the architecture based on IETF MGCP standard. The standard details call flows between different types of users. Based on the call flows, details about messages were extracted from the log files and stored in the data warehouse. In traditional telephony, call set-up delay is relatively simple to compute since a circuit switched network carries signaling over dedicated lines. However, in a packet switched network, where signaling packets can get multiplexed with other traffic this computation is not trivial. Also, the delay has more variance depending on the load in the network. This metric can be derived from the log files by associating the messages within a call context. The information from the logs had to be stored in the warehouse so that variances in call set-up time and hence variances in conformance to SLAs could be identified. The call set-up delay is a typical metric in user SLAs. Although we only looked at log files for call set-up delay, issues in reconstructing call set-up times arose due to the following issues:

- Log files are distributed and messages related to

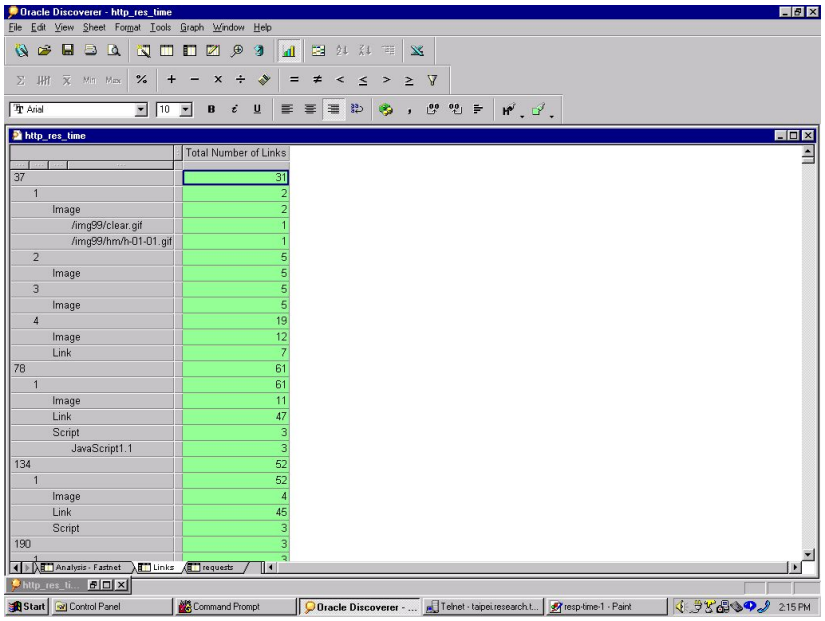

Figure 6: Number of Embedded Links

the same call are stored in multiple places and in different formats. We had to use our collection program and add parsing rules for log files just as we added parsing rules for http traffic.

- Log files are generated as static files and are overwritten after reaching the maximum file size. The copy needed to be placed into the warehouse before the maximum size was reached. We did not have the option of re-naming files and copy them at our convenience.

- Since the time stamps are stored in log files at different locations whose clocks are not synchronized, the time stamps need to be correlated when the logs are moved to the warehouse.

5 Issues and Limitations

5.1 Performance and Distribution

Traffic data can be extremely voluminous. A traditional warehouse approach requires sending all data to a central repository. This introduces problems such as: (1) the transmission of the data adds extra load to the network, and (2) the delay in the transmission results in longer update window of the data warehouse.

We are exploring the idea of implementing a "distributed" data warehouse, in which a number of data warehouses are connected to form a virtual warehouse. A data warehouse would be set up for each network region. All extracted traffic data are directed to and integrated into the regional data warehouses. Only a limited amount of data (e.g. data to correlate IP packets) needs to be exchanged among the data warehouses. Since the data model is homogeneous among all nodes in the network, the same instance of schema can be used on each of the regional data warehouses.

This approach alleviates the problem of data transfer overhead and warehouse update delay. In addition, by harnessing the distributed computing power and storage capacity, it provides a scalable data warehousing architecture in presence of ever increasing traffic. The challenge, then, is how to efficiently process distributed queries and, in particular, OLAP queries.

Within a single warehouse, the IP packet table is the main source of performance bottleneck. We have chosen to cluster IP records on the timestamp attribute. This approach provides efficient performance to queries that analyze the IP traffic in chronological order, but adversely affects queries that analyze on other attributes. For example, consider queries that are related only to HTTP traffic or to the traffic load on a specific subnet. The IP records pertaining to HTTP or a subnet may be scattered throughout the physical table and subsequently incur substantial I/O access. One possible remedy to the problem is to maintain separate IP tables for each application protocol. While this approach provides adequate performance for application-specific queries, it incurs more indexing overhead and is more complicated in schemas and semantics. Another possible solution is to implement the packet table and indices in flat files, with associated operations implemented in specially tailored programs. The advantage is that, with proper implementation, much of the DBMS overhead can be avoided and the performance can be custom-tuned. The

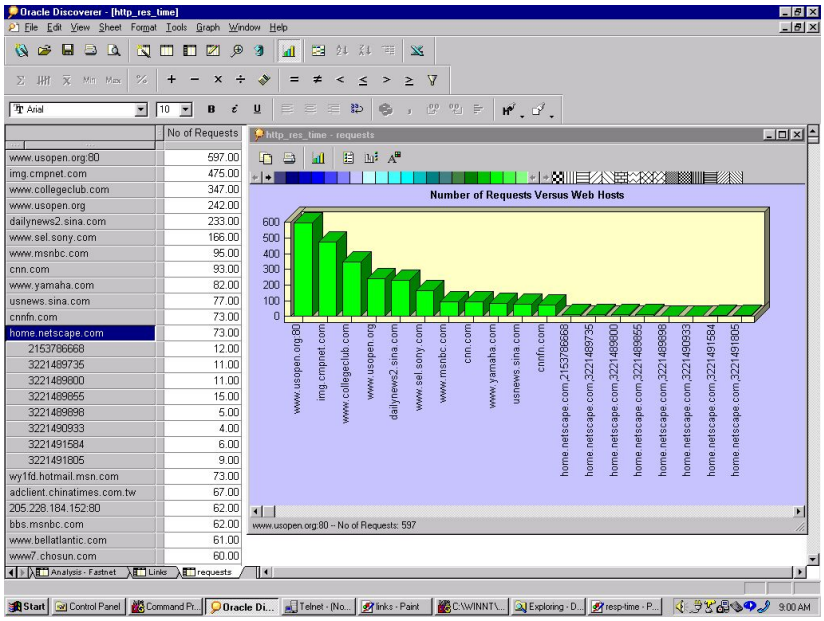

Figure 7: Web site visits

drawback is that we need an interface (APIs) through which we can query the flat files (semi- structured data) and integrate with other data that reside in the relational database. The feasibility of this approach is currently being investigated.

5.2 Historical Warehouse

One feature that is essential to our applications but has not been supported by commercials DBMSs is the incorporation of tertiary storage in the DBMS. Traffic data arrive at a speed that is several magnitudes faster than the data sources of traditional business data warehouses. Thus, secondary storage is saturated and data needs to be moved to a tertiary storage. This requires supports of the following:

- *Storage transparency*: The secondary and tertiary devices should be provided as a seamless storage system to the database users. Except for some configuration set-ups, the DBA or users must be able to run regular SQL queries without worrying about the location of the data across secondary and tertiary storage devices.

- *Efficient migration*: The DBMS must provide quick migration of data between the two-hierarchy storage system. This includes automatic eviction of data from the secondary to tertiary devices as they become "old", and fast retrieval/recovery of data from the tertiary storage to the disks when they are requested. A hybrid indexing technique to index data on both levels of storage is also needed.

- *Aggregates versus detailed data*: Packet data will typically be moved to tertiary storage at the end of a day or week. However, aggregate data derived from the packet data needs to be stored in secondary storage for a much longer time. Rules for deriving and constructing aggregate data as well as dependencies between packet data and aggregate data need to be maintained. The ideal case would be for the DBMS vendor to provide a mechanism for the user to design and build these rules in the warehouse configuration.

5.3 Protocols and Collection

Collection of data and modeling of protocols raise interesting issues for warehouse design. Some of these issues are:

- *Hierarchical structure of Internet protocols*: The layered network protocols suggest a hierarchical data model design. However, the design of the schemas (including base tables and materialized views/aggregates) so as to support efficient vertical drill down (through the protocol stacks) and horizontal traverse (cross-protocol analysis) remains an issue.

- *Data quality*: We have found that the location of the collector in a network can affect the quality of the collected data. Due to asymmetric routing in the Internet backbone, incoming and outgoing traffic may not travel on the same physical link. Collection at a single point in the network could result in observing only one side of a 'conversation'.

To alleviate this problem, we collect near the application to get re-united data. On collections from internal network locations, we preprocess the data to remove incomplete sessions. Further research is needed to deal with incomplete data.

- *What level to collect?* Collecting all the packets enables us to get an in-depth view of the network. Additional information not sent over the network can be gleaned from other sources like application logs such as those generated by web servers, firewalls or other applications. Information about the current state of the network can be obtained by periodically extracting appropriate measurement counters directly from network elements that have SNMP access to their MIBs. More detailed data can be found in the data fields maintained by RMON and RMON2 probes, which extend the data recording capabilities of most network elements. Correlating multiple data sources allows us to develop a more comprehensive picture of the state if the network. Additionally, some of the data source can be used to reduce the size of the data in the warehouse. The proper level and detail of data is an area of further research.

6 Project Status and Future Work

We have built the traffic warehouse and are currently using it to monitor SLAs within our local area network. The project is integrated with a traffic management IP telephony project where packet loss, jitter and delay can be experimentally controlled. We plan to introduce billing and other applications in this experiment to determine the feasibility of a warehouse.

Future work involves exploring issues identified in the paper as well as the following:

- A logical extension to this work involves capturing packet level information at multiple locations in a network. For example, probing at both the application and client processor network segments will enable the calculation of additional SLA parameters such as packet loss and packet delay. The amount of information needed to be stored in the warehouse would not increase substantially, since redundant information is eliminated. Only packets that do not appear in both traces would need to be added. In addition to SLA parameters, multiple correlated traces can be used to determine the location of packet loss in a large network. This can be used to verify or indicate suspected problem sources.

- Our goal is to allow end-users to monitor their own SLA if desired or monitor their own bills. This will require a web front end to end-users where customers can view their own information and aggregate information (patterns of customers like me) but not detailed information of other users.

- End-users are also typically interested in when their services will be activated. This requires access by end-users to service status databases that are not part of the traffic warehouse. This requires integration of the databases into our system, understanding the business processes of service activation, and figuring out which part to expose to end-users.

- We would like to extend the system to include real-time probes and populate the data into the warehouse for real-time network traffic monitoring and analysis. This real-time functionality is coincided in [2], which monitors and analyzes traffic data in traditional telephone networks. Real-time data also facilitate comparisons between historical and current data can be made to understand the state of the network.

References

[1] S. Aidarous, C. Rad, K. Smith, E. Bagnasco, and S. Desai. Service level agreements: can you deliver what you promised? In *Panel Session, Sixth IFIP/IEEE International Symposium on Integrated Network Management*, Boston, MA, May 1999. presentation slides avaialble at www.comsoc.org/confs/im/99/panel.html.

[2] Q. Chen, M. Hsu, and U. Dayal. A data-warehouse/OLAP framework for scalable telecommunication tandem traffic analysis. In *16th Intl. Conf. on Data Engineering*, San Diego, CA, Feb 2000.

[3] A. Erramilli and J. Wang. Monitoring packet traffic levels. In *Proc. IEEE Globecom*, pages 274–280, San Francisco, CA, 1994.

[4] R. Kimball. *The Data Warehouse Toolkit: Practical Techniques for Building Dimensional Data Warehouses.* John Wiley & Sons, 1996.