ABSTRACT

Title of Thesis: CONSTRUCTION OF LATENCY PROFILES : PERFORMANCE MONITORING ON THE PLANET-LAB OVERLAY NETWORK

Degree candidate: Hyma S Murthy

Degree and year: Master of Science, 2004

Thesis directed by: Professor Samrat Bhattacharjee
Department of Electrical and Computer Engineering

One of the prominent challenges in deploying Wide Area Applications (WAA) is scalable performance management. The unpredictable behavior of WAN calls for models to predict the end-to-end latency between a client and server. Early research in this area presents the concept of *Latency Profiles (iLPs)* as a tool to capture the changing latencies experienced by clients when connecting to a server. It also presents a technique to group *iLPs* into *Aggregate Latency Profiles (aLPs)*, study the relationships between *iLPs* using concepts of Mutual Information and Correlation and managing a large collection of *iLPs* using *Relevance Networks*.

Present research in this area, which is also presented in this thesis, deals with a new method of latency prediction using *peers*, apart from using *aLPs* as done earlier. The method involves identifying a group of peer clients experiencing similar latencies to

servers and building vector of *confidence values* in peer clients for each client (for each server). These confidence vectors can be used for latency prediction.

The success of this research on scalable performance monitoring must be validated against thousands of *iLPs*. A new experiment on Planet-Lab is designed for this purpose. PlanetLab is a globally distributed wide area testbed for deploying newtork services at the Internet scale. The experimentation on Planet-lab involves deploying clients and server written in Python. This ensures a better control over the working of clients and servers and also takes care of other details like recording the processor load at both the client and servers and obtaining AS-level BGP paths between clients and servers. The additional parameters of processor loads and BGP paths help to better analyse the relationships between *iLPs*.

CONSTRUCTION OF LATENCY PROFILES: PERFORMANCE MONITORING ON THE PLANET-LAB OVERLAY NETWORK

by

Hyma S Murthy

Thesis submitted to the Faculty of the Graduate School of the
University of Maryland, College Park in partial fulfillment
of the requirements for the degree of
Master of Science
2004

Advisory Committee:

Professor Samrat Bhattacharjee , Chairman
Professor Louiqa Raschid
Professor Ashok Agrawala

©Copyright by

Hyma S Murthy

2004

ACKNOWLEDGEMENTS

I would like to thank Dr. Louiqa Raschid for being a wonderful advisor and giving me an opportunity to work with her, Dr Samrat Bhattacharjee and Dr Ashok Agrawala for their support, cooperation and advice.

I would also like to thank my friends Mukul, Abheek, Akhil, Siddharth and Vinay. This work would not have been possible without their help.

TABLE OF CONTENTS

LIST OF FIGURES

Chapter 1

Introduction

1.1 Wide Area Application and Challenges

Wide area applications (WAAs) utilize a WAN infrastructure (*e.g.*, the Internet) to connect a federation of hundreds of servers, typically content providers, with tens of thousands of clients. Servers provide services that may range from downloads of digital content to authentication sessions, involving two or more parties. It is expected that such applications must scale to tens of millions of resources.

WAAs, while promising in their scope and impact, face significant challenges. A prominent challenge involves the unpredictable behavior of a dynamic WAN [14, 15] that results in a wide variability in access latency (end-to-end delay). In [12], *latency profiles* is proposed as a conceptual modeling tool for the behavior of sources over a WAN. Latency profiles are time-dependent latency distributions that capture the changing latencies clients experience when accessing a server.

Latency profiles [12] can be utilized as a WAA monitoring tool, gathering efficiently data on access and performance patterns. They can also serve to predict latencies that clients should expect in response to requests. Therefore, latency profiles can assist in

personalizing services of WAAs to client's specific network capabilities, including available bandwidth and "distance" from server, in order to improve service delivery in a heterogeneous WAN environment. However, in the presence of hundreds of servers and tens of thousands of clients, managing millions of latency profiles cannot scale. Therefore, mutual information and correlation are used to define latency profile similarity, and use them further to aggregate similar latency profiles. Also to analyze meaningful relationships among *iLP*s is studied using *Relevance Networks* (RN) [4]. RN has been developed for functional genomic clustering to discover non-random associations between genes on the basis of their biological characteristics. Here RN is used for building and maintaining aggregate latency profiles.

An experiment is thus designed on PlanetLab to build latency profiles between client-server pairs, study their relationships and aggregate them into an *aggregate latency profile* using Relevance networks.

1.2 Overview of Planetlab : An Overlay testbed

Planet-Lab [3] is a globally distributed wide area testbed for deploying various network services at the Internet scale. The services experience all the behaviors of the real Internet in terms of paths taken, latency, available bandwidth, connection properties, network presence and geographical location. PlanetLab currently consists of 350 machines, hosted by 150 sites, spanning 20 countries. All the PlanetLab machines run a common software package that includes a Linux-based operating system, mechanism for bootstrapping nodes and distributing software updates, a collection of management tools that monitor node health and facilities for managing user accounts and distributing keys (Public keys). A Public key/ Private key encryption is needed to set up an account on PlanetLab. The objective in PalnetLab is to allocate a slice of of network-wide

resources to an application, allowing it to run across all (or some) of the machines distributed globally.One of PlanetLab's main purposes is to serve as a testbed for overlay networks. A variety of services can be run in the PlanetLab slice like content distribution networks, routing and multicast overlays and network measurement tools. Using an overlay as both a testbed and deployment platform has various advantages like access to a large set of geographically distributed machines, a realistic network substrate that experiences congestion, failures and diverse link behaviors and also a potential for a realistic client workload. Services currently running continuously on PlanetLab include NetBait (a worm detection and tracking device), CoDeeN (a content distribution network), ScriptRoute (a programmable network management service), Chord (a scalable object location service) and Sophia (a network monitoring service). The various advantages of using PlanetLab as an overlay testbed combined with some of the useful network monitoring services running on it (which provide useful information about the BGP paths between any client and server pair and also the continuous ping information between them) make it a very apt platform for running the Latency prediction experiment using client-server Latency profiles.

1.2.1 NIXES ToolSet

Nixes [2] provides a set of bash scripts to install, maintain, control and monitor applications on PlanetLab. It bootstraps the nodes with *yum* and installs the required *rpms* from PlanetLab distribution. Nixes uses *threads* to acheive parallelism. The tool is based on three components: the scripts, a configuration file and a public web repository hosting the application. The advantages of this tool is that it is fast, bootstraps the nodes with yum, gzip and python, deploys any kind of directory structure to the nodes and executes commands on all hosts.

Tools

All the tooles take as argument a list of nodes, where each node is a fully qualified Internet address. The tools use public/private key authentication to log on to the remote nodes via *ssh*. The private key is specified in the *configuration* file.

- *plsetup node-list* bootstraps the vserver running on the nodes specified in the node-list with yum, gzip and python2.3.
- *plinstall "rpms" node-list* installs all the rpms on all the nodes in node-list (rpms must be a part of the redhat distribution)
- *pldeploy node-list* deploys any file structure to the nodes
- *plcmd command node-list* executes any set of commands on all the nodes. *command* refers to an environment variable containing the script to execute. It is specified in the .nixesrc (see CONFIGURATION). The command is either a *START* or *STOP* command.

All the tools work in parallel with by default 40 processes. they write a log file per node basis in the directory specified by the TMP environmen variable. Success of an operation will be notofied by means of no errror, "see the log for more details", in case of failure.

Configuration

The *".nixesrc"* resides in the user's home directory and contains the configuration for all the tools. The plcmd script supports custom options, which specify a script for an action. A sample .nixesrc file is shown below.

Deployment

The deployment script is fully configured within .nixesrc in the home directory. The files are to be stored in a publically available server, and that should be specified in the .nixesrc file.

Control

Any script or command can be executed on all the nodes in parallel. The two commands used are *START* and *STOP*. If the return code is 255, then the control script will echo the output of the command to stdout.

1.3 Related Work

There has been considerable work in the area of distributed performance estimation and monitoring of Internet applications. Commercial solutions which address this issue are Akamai, Keynote, Appliant etc. Some of the many inteeresting research solutions are IDMaps, GNP, RON, Internet Iso-bar, Sophia and Ganglia. Below is a brief description of some of these tools.

IDMaps Internet Distance Measurement Service(IDMaps) [1, 6] is a pioneering work in the area of internet distance measurement. It provides the distance estimate between any two hosts connected to the Internet. It comprises of two components,*Tracers* and *Servers*.*Tracers* measure distances between IP address prefixes and among themselves.*Servers* collect measurement results and answer distance queries. The distances between two hosts, A and B is the sum of *three* distances, distance from A to *Tracer* $T1$, distance B to *Tracer* $T2$, and the shortest distance between $T1$ and $T2$. Traceroute is used to know the path between the hosts. Scalability issue is addressed by grouping the IP addresses into prefixes.

Global Network Positioning - *GNP* This approach models the Internet as a geometric space and computes geometric coordinates to characterize the position of hosts in the internet. A key feature of mapping the hosts in the geometric space is identifying a small distributed set of cooperating hosts called *Landmarks*. The *Landmarks* compute their own distances in a chosen geometric space and disseminate to hosts wanting to compute their own coordinates relatively in the geometric space. Network distances between the hosts are then predicted by evaluating a *distance function* (like Euclidean distance) over their coordinates.

Internet Iso-bar Internet Iso-bar [5] is a scalable overlay distance monitoring system. It clusters hosts based on the similarity of their perceived netwrok distance. The centers of these clusters are then used for continous probing and distance estimation. Given a set of N end hosts that belong to different administrative domains, a subset of those nodes is selected and an overlay monitoring system is built without any knowledge of the underlying topology. It is a real-time monitoring system, easy to implement, scalable and has small communication and computation cost. Various distance correaltion metrics are used to determine the *correlation distance* between pairs of hosts denoted by $cor\_dist(*,*)$, such as Network distance correlation, Network similarity in terms of Euclidean distance between network distance vector and vector similarity between network distance vectors and correlation based on Geographical proximity. Clustering is based on two methods, the *K-center problem*, which runs in $O(N^3)$ time, and on *minimum set cover problem*. The internet iso-bar can also be used to measure distance between two hosts in a peer-to-peer system and between a client c and a server s. The performace of iso-bar is compared with Global Network Positioning (*GNP*) and the results show that there is a negligible difference between the two, even though *GNP* shows a slightly better performance than iso-bar.

Sophia Sophia [16] is a distributed system that collects information about the various elements in the network, evaluates statements and reacts according to the evaluations of the information collected. The components of Sophia include *sensors* to collect information about the network elements, *declarative programming environment* to evaluate logic statements regarding the system and *actuators*, which perform the local actions. Explicit notions of time and location are embedded into Sophia making it easy to distribute expressions across the network and process both past and future information. The interface to Sophia core is the *eval(Term)* functor. Metrics involving netwrok parameters like bandwith and latency and logic expressions involving them can be specified as predicates. The actual values associated with the predicates are returned by the sensors running on nodes throughout the network. Evaluation results including the facts from the sensors are cached at each node which helps in situations where computation latency is more important than freshness of data. The other design aspects of Sophia include *Pre-Scheduling*, *Evaluation planning*, which is analogous to query planning and optimization in databases, *Extensibility* in terms of adding functionality over time and using *Capabilities* to assign and enforce privileges and for module protection. The implemetation of Sophia includes a *logic terms database* for storing all Terms, a *logic unification engine* based on standard logic unification, interfaces with sensors and actuators which form the I/O of Sophia, *remote evaluator* for delegating evaluation of an expression to a particular remote node and an *expression scheduling mechanism* to maintain the calendar of evaluations scheduled for the future.

Ganglia Ganglia [10] is a scalable distributed monitoring system for high performance computing systems like clusters and Grids. It is a hierarchical design targeted at federations of clusters. It uses a multicast listen/announce protocol to monitor state within clusters and uses a tree point-to-point connections among representative cluster

nodes to aggregate state information from the clusters. It uses XML for data representation, XDR for compact portable data transport and RRDtool for data storage and visualization. Within each cluster, Ganglia uses heartbeat messages on a well-known multicast address as the basis for a membership protocol. Each node monitors its local data and sends the updates on the multicast address. All nodes listen for data on this address and thus have data about all the other nodes in the cluster. Ganglia federates multiple clusters using a tree of point-to-point connections. Each leaf node specifies one node in a cluster and points higher up in the tree specify aggregation points. The monitoring data is represented in XML. Ganglia has been implemented for clusters, grids and on Planet-Lab. Experimentation on planet-lab has revealed several shortcomings of Ganglia in terms of assumptions of cheap wide-area bandwidth, large I/O overhead, lack of hierarchical namespace, lack of timeouts on monitoring data and scalability. Some of these shortcomings are said to have been modified in the newer versions.

Chapter 2

Wide Area Performance Monitoring

In this chapter latency profiles [12, 13] is discussed as a conceptual model of client-server interaction. Latency information between a client and server is maintained in form of latency profiles. We apply passive information gathering strategies for building individual latency profiles (*iLP*s) for client/server pairs. However, managing millions of individual latency profiles does not scale. Below we consider a method for aggregating latency profiles that improves scalability of the WAA monitoring.

The main idea of the approach is as follows. Suppose that a client/server pair (c,s) does not have an associated *iLP* that can be directly used to optimize access from c to s, or alternatively, the system does not have sufficient resources to continuously maintain such a profile. Assume further that there is a well-defined individual latency profile iLP_1 associated with a client/server pair (c_1,s). In addition there are latency profiles iLP_2 and iLP_3 associated with client/server pairs (c,s_1) and (c_1,s_1). If there is a non-random association between iLP_1, iLP_2 and iLP_3, we argue that a reasonable estimate of access latency for (c,s) can be obtained by grouping iLP_1, iLP_2 and iLP_3 in an aggregate latency profile *aLP*.

The general approach is to construct *iLP*s and aggregate them to improve the overall

prediction power.

The rest of this chapter gives a formal definition of *iLP*, grouping of similar (non-randomly associated) *iLP*s in aggregate latency profiles, introduce relevance networks as a tool for constructing aggregate latency profiles amd how aggregate latency profiles can be used for predicting latencies.

2.1 Individual Latency Profiles

Given a client c, a server s, an object of size b, and a temporal domain T, an *individual latency profile* is a function $iLP_{c,s} : T \times b \rightarrow \Re^{+} \cup \{TO\}$. *iLP* represents the end-to-end delay for a request from server s at time t. TO represents timeouts. $iLP_{c,s}$ comes in two flavors, similar to [8]. One flavor measures time-to-first, which depends on factors such as workload at the server and size of the requested object. The other flavor measures time-to-last, which has a greater dependency on network bounds.

Due to the stochastic nature of the network, $iLP_{c,s}(t)$ is clearly a random variable, yet its specific representation can vary. Below assume $iLP_{c,s}(t) = iLP_{c,s}$ for all t, to be a discrete time-independent random variable, represented as an $\begin{pmatrix} L \\ p \end{pmatrix}$ matrix where $[L] = [L_1, L_2, ..., L_n]$ is a row matrix of latencies and $[p] = [p_1, p_2, ..., p_n]$ is a row matrix of corresponding latency probabilities ($\sum_{k=1}^{n} p_i = 1$).

Example 2.1.1 *As an example consider the following probability distributions corresponding to two individual latency profiles (X and Y represent specific client/server pairs):*

$$LD_X = \begin{pmatrix} 1 & 2 \\ 0.5 & 0.5 \end{pmatrix}, LD_Y = \begin{pmatrix} 2 & 3 \\ 0.75 & 0.25 \end{pmatrix}$$ □

2.2 Aggregate Latency Profile

An *aggregate latency profile* aLP_{LP} combines a set of n individual latency profiles $iLP = \{LP_{c_i,s_i}\}_{i=1}^{n}$. Constructing an *aLP* involves grouping *iLP*s with similar characteristics in order to improve overall latency prediction. Apparently, to achieve improvement in the prediction quality one has to aggregate only *iLP*s that are non-randomly associated with each other. Therefore, one should have a methodology to evaluate candidate *iLP*s for aggregation purposes. Below is an introduction to the concept of LP similarity that can be used to provide such evaluation.

We define a *similarity function* $\Sigma : CS \times CS \times \mathcal{T} \rightarrow SM$, where CS is the set of all possible client/server pairs, $\mathcal{T}$ is a set of finite time regions (possibly intervals), and SM will be discussed shortly. Σ is a function that measures, given two latency profiles, their similarity over $\tau \in \mathcal{T}$.

There are two specific measures of latency profile similarity based on mutual information [7] and correlation [11]. To estimate the similarity of two *iLP*s, we consider their joint behavior described by a joint probability matrix $[P(X,Y)]$. $[P(X,Y)]$ provides the probabilities of the joint occurrence of two latencies.

Example 2.2.1 *Consider X and Y, given in Example 2.1.1. Their joint probability distribution is given as:*

$$P(X,Y) = \begin{pmatrix} (1,2) & (1,3) & (2,2) & (2,3) \\ 0.5 & 0 & 0.25 & 0.25 \end{pmatrix} \qquad \square$$

Mutual information between two random variables $MI(X,Y)$ is defined as

$$MI(X,Y) = \sum_{i,j} \left(p_{i,j} \lg \frac{p_{i,j}}{p_i p_j} \right) \tag{2.1}$$

where $p_{i,j}, p_i, p_j$ are joint and individual probabilities of the latencies X and Y, respectively. A higher mutual information between two *iLP*s means that those *iLP*s are non-

randomly associated. Conversely, a mutual information of zero means that the join distribution of *iLP*s holds no more information than their individual distributions.

We define correlation between two random variables $Corr(X,Y)$ as follows:

$$Corr(X,Y) = \frac{1}{n-1}\sum_{i,j}\left(\frac{x_i-\bar{X}}{S_X}\right)\left(\frac{y_i-\bar{Y}}{S_Y}\right) \tag{2.2}$$

The correlation coefficient as defined above measures the degree of the linear association between two variables. A higher correlation between two *iLP*s can also indicate that those *iLP*s are non-randomly associated. In general, there is no straightforward relationship between correlation and MI [9]. While correlation captures linear dependence, mutual information is a general dependence measure.

Example 2.2.2 *For the latency distributions of Example 2.1.1 and Example 2.2.1 we calculate mutual information* $MI(X,Y)$*, as follows:*

$p_{1,2}\lg\frac{p_{1,2}}{p_1p_2} + p_{1,3}\lg\frac{p_{1,3}}{p_1p_3} + p_{2,2}\lg\frac{p_{2,2}}{p_2p_2} + p_{2,3}\lg\frac{p_{2,3}}{p_2p_3} = 0.31$

As for correlation, $\bar{X} = 1.5, \bar{Y} = 2.25, S_X = 0.58, S_Y = 0.5$*, and* $Corr(X,Y) = 0.57$.

□

2.3 Constructing Aggregate Latency Profiles using Relevance Networks

This section proposes an approach to analyze and visualize meaningful relationships among *iLP*s using Relevance Networks (RN) [4]. RN has been developed for functional genomic clustering to reveal non-random associations between genes on the basis of their biological characteristics. In this research RN is applied in the context of WAA

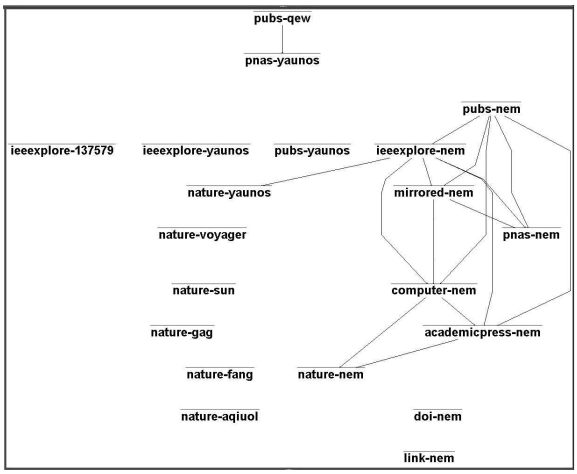

Figure 2.1: Relevance network example

performance monitoring. In particular, how RN can be used for building and maintaining aggregate latency profiles.

The RN-methodology is based on computing pair-wise relationships (*e.g.*, correlation and mutual information) for all *iLP* pairs. Consider a graph whose nodes represent *iLP*s and edges represent the relationships (associations) between them. Assume that we compute all pair-wise relationships. By choosing a relationship threshold and displaying only those edges with a relationship higher then the threshold, then, out of completely connected network of *iLP*s, we extract clusters of *iLP*s whose relationship to each other is "stronger" than the threshold. Such clusters are called Relevance Networks. Observing how the threshold increase impacts characteristics of the Relevance Networks (*e.g.*, number of edges and number of connected components), one can generate a set of *iLP* relationships and aggregate strongly related *iLP*s. One outcome of this approach is that it provides us with a natural quality estimation of both *aLP*s (aggregate only above certain threshold), and the whole group of candidate LPs (sensitivity to threshold increase).

Example 2.3.1 *Figure 2.1 provides an example of a relevance network, as was generated during the experiments. Each node is a client-server pair (*e.g.*,* `pubs-qew`*) and an*

edge between two nodes represent a similarity above the network threshold. The RN in this example has two connected component (one of size 2 at the top of the figure and the other of size 8 at the right hand side of the figure. □

To build a Relevance Network, one needs an input feed in the form of $\langle iLP_1, iLP_2, Measure \rangle$ and a threshold specification. The measure, in our case,is either the correlation or mutual information between iLP_1 and iLP_2.

2.4 Latency Prediction Using Aggregate Latency Profiles

After constructing an *aLP* from a set of *iLP*s, one can improve prediction quality of an *iLP* using observations of *iLP*s from the same *aLP*. To demonstrate that the meaningful relationships between profiles within an *aLP* discovered in the previous section can be used to improve the quality of latency prediction we will use latency estimations using conditional expectation (CE).

A well-known fact from estimation theory is that the expected value $E(X)$ of a random variable X minimizes the expected value of the mean-square-error of estimation $E\left((X - est_X)^2\right)$ [11]. Using an observation of a second random variable Y which is related to X in some way (*e.g.*, Y is correlated with X), an optimal mean-square-error estimator of X given Y is the conditional expectation $E(X|Y)$ of X given Y [11]:

$$est_X = E(X|Y) = \sum_{x_i} \left(x_i p(x_i|y_i)\right) \tag{2.3}$$

where $p(x_i|y_i)$ is the conditional probability of x_i given y_i, which can be easily calculated

from the joint probability distribution $p_{i,i}$, using the following equation:

$$p(x_i|y_i) = \frac{p_{i,i}}{p(y_i)} \tag{2.4}$$

Example 2.4.1 *Using latency distribution from Example 2.1.1 and Example 2.2.1, we calculate the following conditional probability distribution:*

$$P(X|Y) = \begin{pmatrix} 1|2 & 1|3 & 2|2 & 2|3 \\ 0.67 & 0 & 0.33 & 1 \end{pmatrix} \tag{2.5}$$

Then,

$E(X|Y=2) = 1*0.67+2*0.33 = 1.33$

$E(X|Y=3) = 1*0+2*1 = 2.$

*It is obvious that conditional expectation based estimation outperform estimation based on simple expectation, which in this case would be E(X) = 1*0.5+2*0.5=1.5.*

Chapter 3

Experiments

This chapter discusses piror work done in building Latency Profiles, Analysis done on the data collected and setting up of an experimentation on a larger scale on Planet-Lab.

3.1 Prior Work

The prior work in using Latency Profiles as a tool for Performance Monitoring of Wide area Applications involved building Latency Profiles, aggregating them and using the *aggregated LP* in Latency Prediction. Each step will be described briefly below.

3.1.1 Construction of Latency Profiles

A small scale experiment was conducted between August and December 2002 on the CNRI handle testbed and involved simulating a Wide Area Application accessing handles maintained by the International Digital Object Identifier (DOI) Foundation (www.doi.org). The Handle protocol is an emerging $IETF/IRTF$ standard providing a global name service for use over WANs, a namespace, a name resolution service and protocols for digital object location and access.

The data was *PDF* files got by resolving the handle_ids via the Handle Protocol. Data objects of size varying from 70 to 100 KBytes were identified at the ten most popular *content servers*.

The location of clients was dictated by their accessibility. The **Clients** therefore included 4 to 8 Handle clients placed in University ASes located across *Europe, North America* and *Australia*. The JAVA Handle client deployed, periodically resolves a group of Handles and downloads the corresponding *PDF* files using $HTTP$ requests. The latency logs of obtaining the files, latency being the sum of Handle resolution time and the time to contact the repository and downloading the files, was sent to the data analysis site at the *University of Maryland* via email using JAVA SMTP libraries.

Data with respect to 22 clients (2 each on 11 client ASes) accessing 10 servers, yielding 22 *iLPs* was studied. For each pair-wise combination of 2 *iLPs*, data processing was performed in terms of *Alignment* (based on Timestamps and a minimum granularity of one hour) and *Normalization* (for statistical comparison). Further *Similarity computation* was carried out, the measures being **Mutual Information** and **Correlation**. These similarity measures were then used to identify connected components in *Relevance Networks*.

3.2 Analysis: Buiding Peer Confidence Vectors for Latency Prediction

One way to predict latency of a client is to use another *peer* client as a predictor, a peer in whose latency the client maintains a high confidence. This involves each client maintaining a *Vector* of confidence values with regard to other *peers* as *predictors*. The steps followed in building such a vector along with the mathematical formulae used is

described below.

Step 1: Identify *peers*(clients), who share similar latency values and group them for peer prediction. The experiment conducted earlier on which this analysis is to be carried out involved 22 clients. Due to the small number of clients, each client considers all the remaining 21 other clients and builds the confidence vector for them.

Step 2: Define the Confidence Vector. The confidence vector ***Conf_Vect*** is built both at the day-hour level and at the day level for each server. It is a two dimensional vector having m rows (corresponding to the days of the week, ranging from 0..7) and n columns (corresponding to the hours of the day, ranging from 0..24). Each element of the vector ***Conf_Vect[d][h]*** is a vector of length 21 and stores the confidence values the client has in the other clients for that day and hour. Similarly the ***Conf_Day_Vect*** is a d dimensional vector (corresponding to the days of the week, ranging from 0..7) and each element of this day vector is a vector of 21 values, again corresponding to the confidence values the client has in the other clients for that day. The data structures for the vectors is shown in figure 3.1 and figure 3.2 respectively .

Step 3: Building the confidence values. Two confidence parameters are introduced here, ***Instantaneous Confidence*** and ***Aggregated Confidence***.

Definition: Instantaneous Confidence □

It is defined as the percentage confidence that a particular client has in the ***predicted latency*** (value returned by a peer client) when compared with its own ***correct latency***. It is defined by $S(t)$.

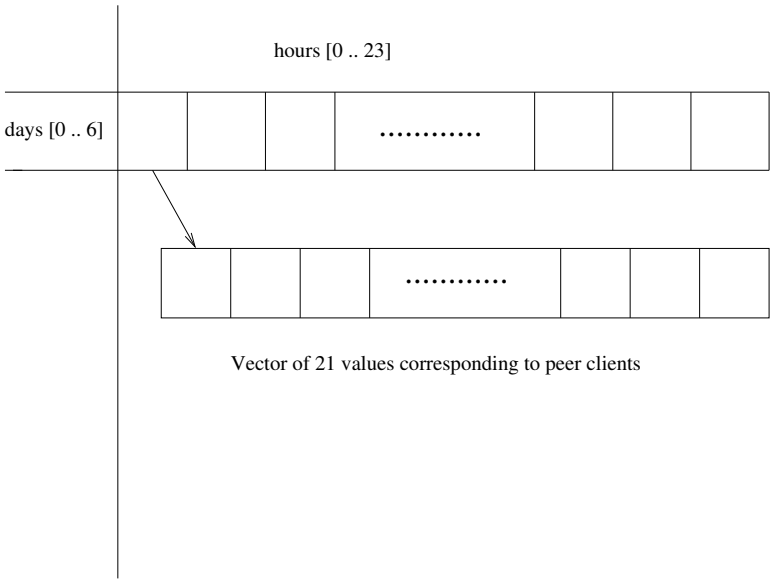

Figure 3.1: Data Structure of Conf_Vect

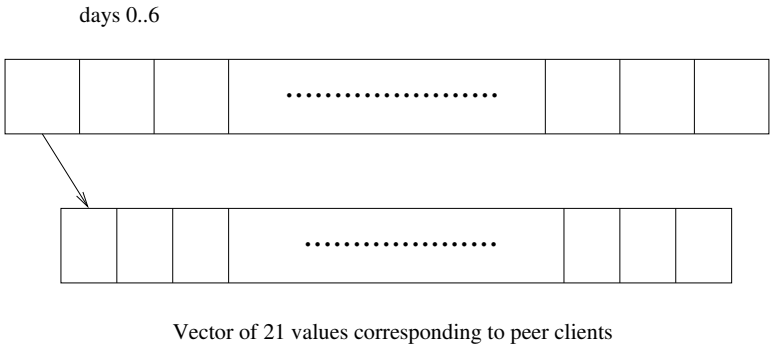

Figure 3.2: Data Structure of Conf_Day_Vect

$$\eta = |predicted - true| \tag{3.1}$$

$$100 \cdot e^{(-\eta/\sigma)^2} = S(t) \quad \sigma = 500 \tag{3.2}$$

Definition: Aggregated Confidence □

It is defined as the *aggregated* confidence a client will have in the peer client's ***predicted latencies*** over a period of time (could be over the entire data set, over a particular day of the week, etc). It is defined by $S(a)_0^t$.

$$S(a)_0^t = \sum_{k=0}^{t} \frac{S(t) \cdot \alpha^{k\,t}}{\sum_{k=0} \alpha^k} \quad 0 < \alpha < 1 \tag{3.3}$$

$$S(a)_0^t = \frac{S(t) + \alpha \cdot S(a)_0^{t-1}}{1 + \alpha} \tag{3.4}$$

The factor α controls the weightage given to previous values. It ranges from 0 to 1, when equal to 1, all values are given equal importance and the aggregated confidence *Sa* will be the average of all the instantaneous confidences over the time.

The data set is organized as follows. There is a separate directory for each server. Each server directory consists of the log files for all the clients accessing it. The log files contain the time of access (Day Date Month Hour:Minute:Second Year) and latency values. Ideally the data set should contain one value for every hour and values for all days of the experiment. However, the handle client timed out quite often resulting in multiple downloads happening in the same hour on some

days and no downloads happening on some other days. Since the data set for analysis is small, to get better analysis results all the latency values of an hour were considered. Hence for a particular date and hour, the latency is a group of values, instead of just one value.

Step 4: Consider the building of the confidence vectors for a client *A*. *A* is interested in obtaining latency values for a particular server *X* for say, ***Friday 12 noon*** and day ***Friday***. In the data collected for *A*, Friday falls on three dates, 06, 13 and 20 of December 2002.

Step 5: Building the Conf_Vect[d][h] for Friday 12 noon

- *A* contructs a *temp_vector* with the latency values for 12 noon on these dates. It then calculates the Instantaneous confidence values for each of the other clients by passing the *temp_vector*.
- At each of the other clients : Lets consider client *B*. For every element of the *temp_vector*, *B* looks up the corresponding (day,date,hour) latency values. If the data is available then *B* calls the function to calculate the instantaneous value. Suppose *A* has 5 values for the hour and *B* has 5. For each of the (5*5) values the ***Instantaneous Confidence*** formula is applied and the average of the 25 values is taken as the Instantaneous value for that (day,date,hour). In case the client *B* has no data collected for that time period, the previous data of *B* is used to calculate the instantaneous values.
- Instantaneous values for all the friday dates are then calculated and the ***Aggregated Confidence*** value is calculated over them. This is the confidence that client *A* will have in client *B* for Friday 12 noon. Similarly the values are calculated for all the other clients.

- The above steps are applied for all the days*hours cycle and the values are entered in *A*'s ***Conf_Vect[m][n]***

  The above steps are better explained in the figure 3.3.

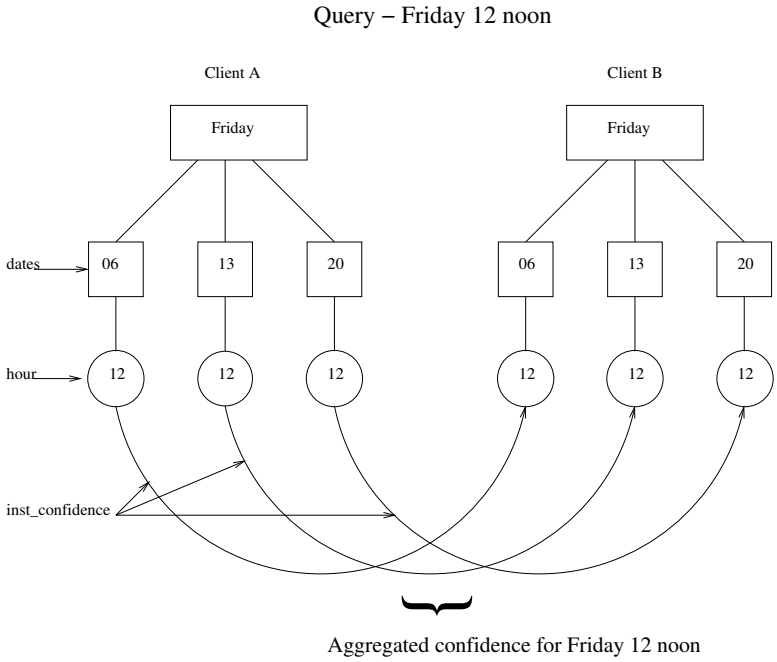

Figure 3.3: Conf_Vect example for Friday 12 noon

Step 6: Calculating the ***Conf_Day_Vect*** values.

- Suppose *A* is interested in building the confidence vector for day ***Friday***. A *temp_vector* is constructed for the dates on which friday falls in the data set of *A*. Each element of the temp_vector will correspond to a date, the hours and the hour latency values. This vector is passed on to the other clients to calculate the instantaneous confidence values for each date.
- At each of the other clients : Consider client *B*. For every element of the *temp_vector*, *B* looks for the corresponding (day, date, hours, hour_latencies). If the data is available, instantaneous confidence values are calculated for all

the hours, *aggregated* to obtain the Instantaneous confidence *A* will have in *B* for that date. Similarly instantaneous values are calculated for all the dates in the temp_vector. In case data is unavailable for *B* for a particular date, previous date's data of *B* will be considered to calculate the instantaneous confidence for that missing date.

- The Instantaneous values obtained above will be aggregated using *Aggregated Confidence* formula to obtain the aggregated confidence *A* will have in *B* for Friday.

  The figure 3.4 better explains the construction of the Conf_Day_Vect.

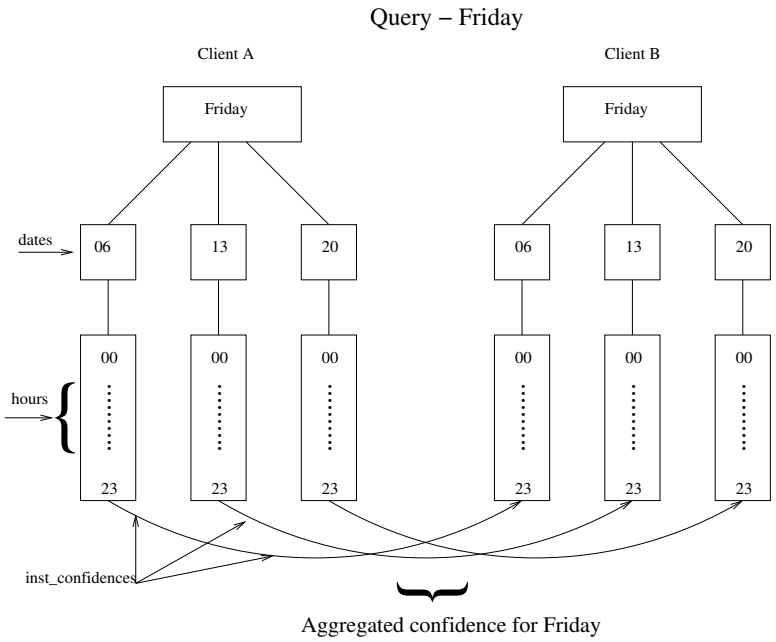

Figure 3.4: Conf_Day_Vect example for Friday

- Graphs are then plotted to show the distribution of the confidence values for each client for each of the servers and over the entire dataset. The example of one such plot for client *UMD-1* to server *APress* is shown in figure 3.5.

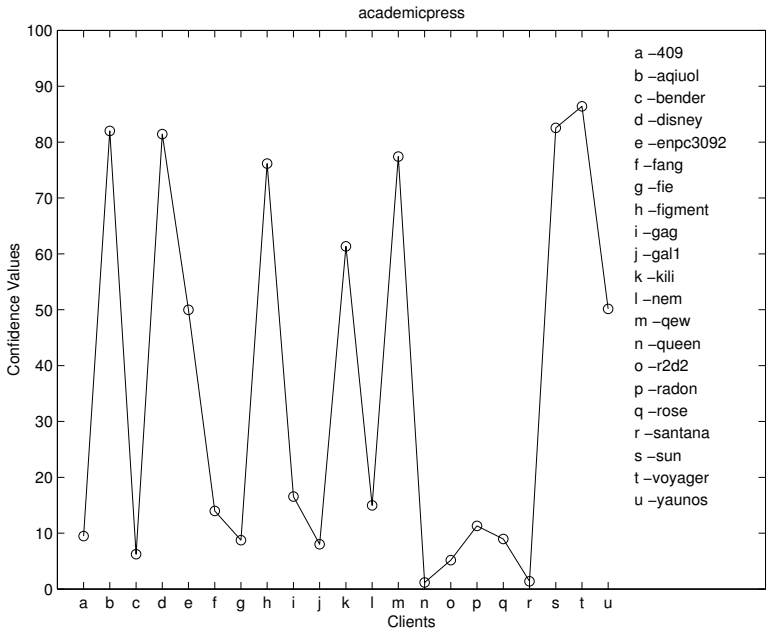

Figure 3.5: Confidence values of client UMD-1 for server APress

3.3 Experimentation on Planet-Lab

3.3.1 Motivation

The experiment conducted earlier during 2002-2003 yielded interesting results based on a small data set of 220 *Client-Server* pairs (220 *iLPs*). However the success of a scalable performance monitoring system must be set against a resonable size data set consisting of atleast a 1000 *Client-Server* pairs. Access to such a large number of clients and servers is only possible with an overlay testbed like Planet-Lab. The globally distributed nature of Planet-Lab provides an ideal platform for deploying large scale network experiments with the experience and perspective of a real Internet, both in terms of geographic location and connection properties. The experimentation on Planet-lab involves deploying our own clients and server written in Python. This ensures a better control over the working of clients and servers and also takes care of other details

like recording the processor loads at both the client and servers. Also by deploying *Sciproute* (a tool by University of Washington), one can obtain the AS-level traceroute information from servers to clients. This helps in further analysing the relationships between the *iLPs* in terms of BGP paths.

3.3.2 Experimental setup

1 Create a slice on Planet-Lab.

2 Choose 40 client nodes and 40 server nodes to add to the slice. The planet-lab sites usually host more than one node. The planet-lab sites are chosen such that each of them have atleast two free nodes and among these nodes, one of them is added as the server node and the other as client node. Thus around 40 planet-lab sites are chosen.

3 A central node is chosen as the Administrative node to install the appropriate packages nodes, deploy necessary files, start and stop processes on all the nodes. "Nixes" is used to perform the above operations at the planet-lab nodes from the central node.

4 Working of the ***Client*** : The client is a *python* client, operating either in the active mode or the sleep mode. During the active session, the client contacts a list of servers and downloads a certain file of size 178KBytes. The client maintains a *log* file for each server it contacts. Each time it contacts the server, it appends to the respective log file, the time-stamp, time for the first set of bytes to arrive (TTF), the total download time (DL), the processor load at the server (SERVER_LOAD) and the processor load at the client at the time of accessing the server (CLIENT_LOAD). The client also maintains an *exception* file for each

server and makes an entry in it in the case of server down or time-outs.

5 Working of the ***Server*** : The server is a *python* server continously accepting client connections and servicing their requests for file transfer. For each connection by a client, the server opens a separate thread to service the client. It also maintains a *log* file for each client that contacts it, and appends the file with the time the client contacted it, the transfer time (TRANSFER_TIME) and its processor load at that time (PROCESSOR_LOAD). The server also runs *scriptroute-rockettrace* periodically to obtain the *BGPRoutes* to the list of clients which contact it.

6 Working of the ***Central Node*** : The central node controls the application running on the planet-lab nodes. It uses the Nixes tool to perform the following operations:

- Bootstrap the nodes with *yum*, *python2.3* and *scriptroute* among other tools. *"plsetup"* command is used for this.
- Deploy the necessary files at the client and server nodes using the *"pldeploy"* command.
- Start the Client and Server programs at the respective nodes using *"plcmd START"* command.
- The central node collects the log files from the clients and servers each day. It also starts rockettrace at the server nodes once a day and collects the output from them. The central node processes the rockettrace output from the server nodes and builds a file for that server for that day containing all the AS-level paths to the clients.

  The client and server planet-lab nodes along with the BGPRoutes are shown in figures 3.6, 3.7, 3.8.

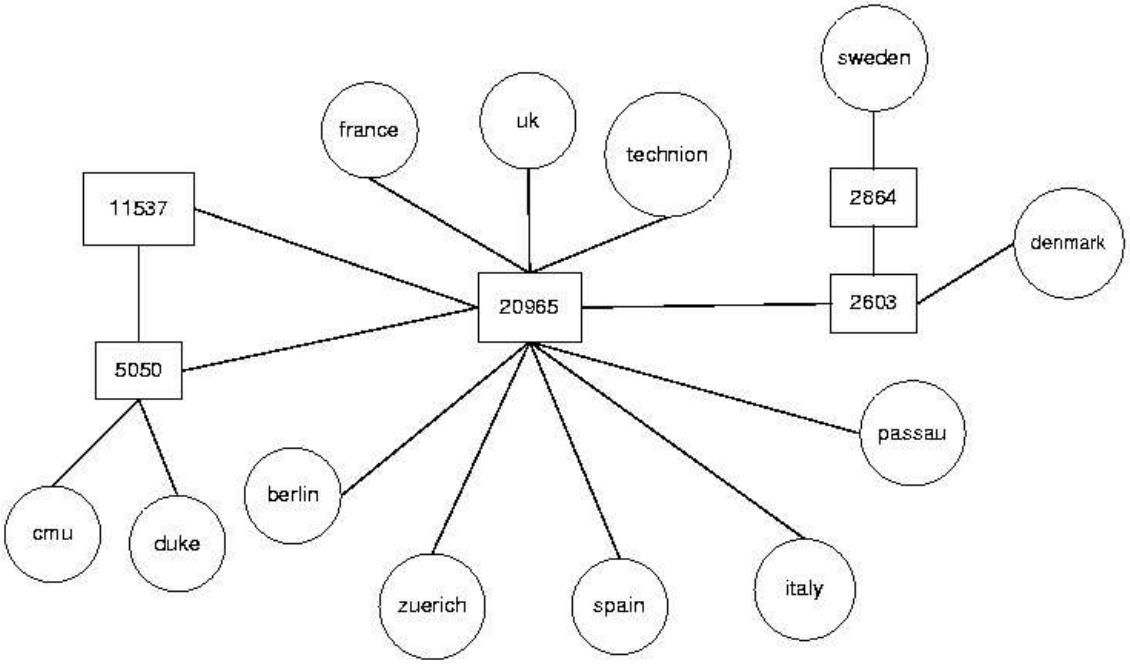

Figure 3.6: PlanetLab Layout 1

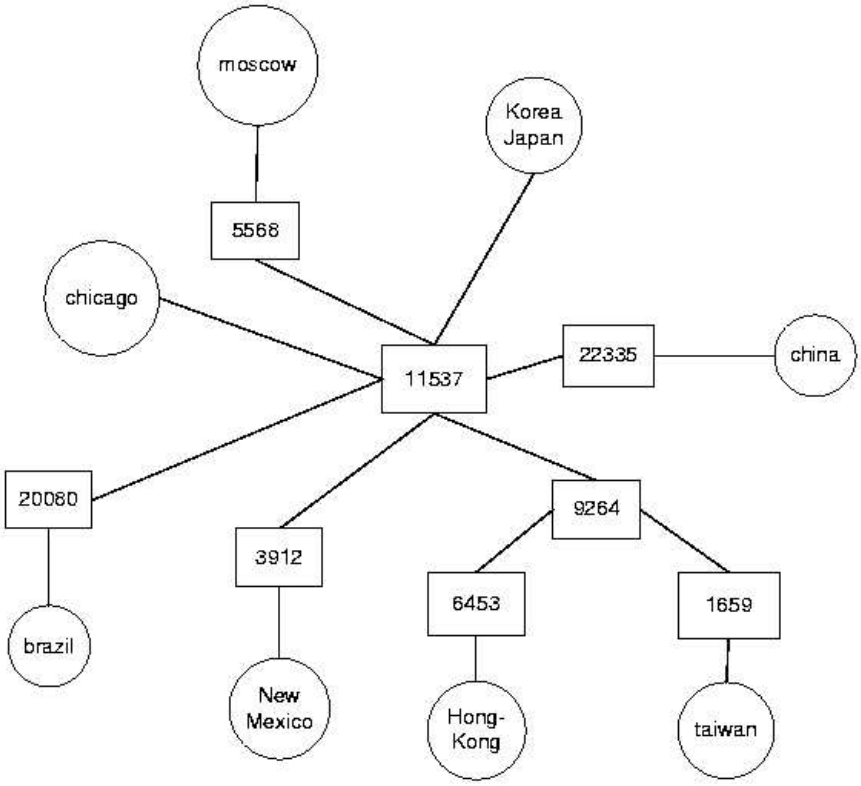

Figure 3.7: PlanetLab Layout 2

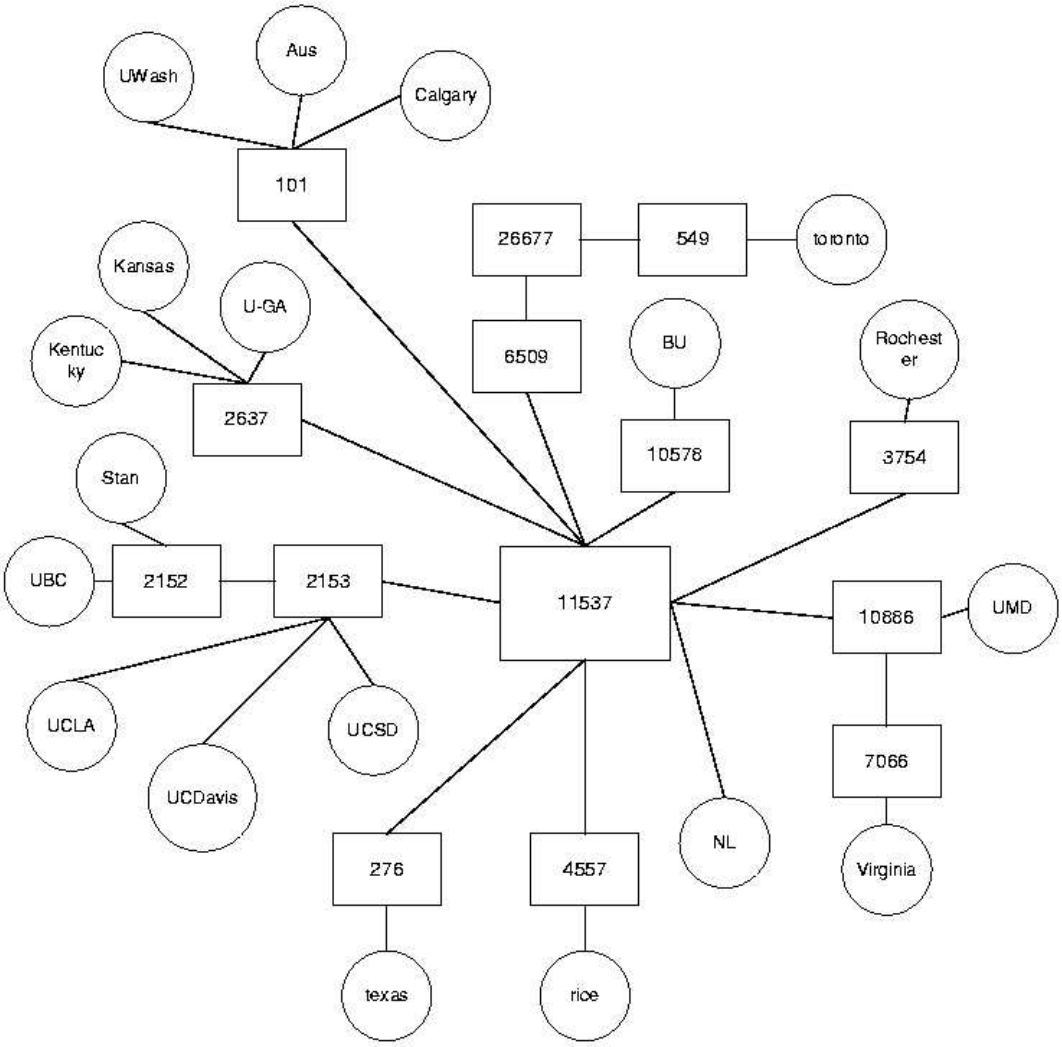

Figure 3.8: PlanetLab Layout 3

Chapter 4

Conclusions and Future Work

This research presents the concept of building and maintaining *Latency Profiles* (*iLPs*) as a method for capturing the behavior of client-server interaction over a WAN. It also proposes a method of *aggregating* the latency profiles into an *Aggregate latency profile* for scalable utilization of latency profiles for latency prediction. Mutual information (MI), Correlation and Building Peer confidence vectors are used to study the relationships between the latency profiles and visualize these relationships by means of Relevance networks.

Even though the analysis of earlier data set yielded interesting results, it cannot be validated against thousands of *iLPs*. The placement of clients and servers was dictated by availability and deployment issues. Hence a new experiment was designed on Planet-Lab, a globally distributed wide area testbed for deploying network services at the Internet scale.

The experiment involves deploying 40 clients and 40 servers at 40 sites of Planet-Lab, yielding 1600 client-server pairs, hence 1600 *iLPs* to study. Also parameters like server processor load, client processor load and BGP paths between clients and servers are looked into in this new experiment. As a result, MI and correlation between *iLPs*

can now be studied against BGP paths. Relationships between clients sharing different lengths of common paths to servers, different client groups against each other can be looked into.New protocols for online peer confidence buildup, peer-to-peer setup for sharing latency values can also be studied.

BIBLIOGRAPHY

[1] http://idmaps.eecs.umich.edu/.

[2] http://www.aqualab.cs.northwestern.edu.

[3] www.planet-lab.org.

[4] A.J. Butte and I.S. Kohane. Mutual information relevance networks: Functional genomic clustering using pairwise entropy measurements. In *Proc. Pacific Symposium on Biocomputing*, 2000.

[5] Yan Chen, Randy Katz, and Chris Overton. Internet iso-bar: A scalable overlay distance monitoring system. In *Proceedings of ACM SIGMETRICS Performance Evaluation Review*. ACM Press, September 2002.

[6] P. Francis, S. Jamin, C. Jin, Y. Jin, V. Paxson, D. Raz, Y. Shavitt, and L. Zhang. Idmaps: A global internet host distance estimation service, 2000.

[7] F.Reza. *An Introduction to Information Theory*. McGraw-Hill, 1961.

[8] J.-R. Gruser, L. Raschid, V. Zadorozhny, and T. Zhan. Learning response time for websources using query feedback and application in query optimization. *VLDB Journal*, 9(1):18–37, 2000.

[9] W. Li. Mutual information functions versus correlation functions. *Journal of Statistical Physics*, (60), 1990.

[10] Matthew L. Massie, brent N. Chun, and David Culler. The ganglia distributed monitoring system: Design, implementation, and experience, 2003.

[11] W. Mendenhall and T. Sincich. *Statistics for Engineering and the Sciences*. Macmillan Publishing, 1985.

[12] L. Raschid, H.-F. Wen, A. Gal, and V. Zadorozhny. Latency profiles: Performance monitoring for wide area applications. In *Proceedings of the Third IEEE Workshop on Internet Applications (WIAPP '03)*, San Jose, CA, June 2003.

[13] L. Raschid, Qiang Ye, A. Gal, and V. Zadorozhny. Latency prediction using aggregate latency profiles. 2004.

[14] D. Rubenstein, J. Kurose, and D. Towsley. Detecting shared congestion of flows via end-to-end measurement. *Proceedings of the ACM SIGMETRICS Conference*, 2000.

[15] M. Stemm, S. Seshan, and R. Katz. A network measurement architecture for adaptive applications. In *Proceedings of IEEE InfoComm*, 2000.

[16] Mike Wawrzoniak, Larry Peterson, and Timothy Roscoe. Sophia: An information plane for networked systems. In *Proceedings of ACM SIGCOMM Computer Communication Review*. ACM Press, January 2004.