

Analysis and Implementation of Relational Archive Site for PingER and CBG Integration with TULIP

BY

Ghulam Nabi 2007-NUST-BIT-108

M. Farhan Maqsood 2007-NUST-BIT-35

Bilal Naseer 2007-NUST-BIT-107



**A Project report submitted in partial fulfillment
Of the requirement for the degree of
Bachelors of Information Technology**

**NUST School of Electrical Engineering & Computer Science National
University of Sciences & Technology**

Islamabad, Pakistan

2011

Certificate

It is certified that the contents and form of thesis entitled “**Analysis and Implementation of Relational Archive Site for PingER and CBG Integration with TULIP**” submitted by Ghulam Nabi, Farhan Maqsood and Bilal Naseer has been found satisfactory for the requirement of the degree.

Advisor:

Dr. Anjum Naveed

Co-Advisor:

Dr. Zahid Anwar

Dedication

We dedicate this work to our parents, who always prayed tirelessly for our success, encouraged us and always gave more than what we deserved!

Surely they are the pillars of strength in our lives.

Acknowledgements

First and foremost, we are immensely thankful to Almighty Allah for letting us pursue and fulfill our dreams. Nothing could have been possible without His blessings. We would like to thank our families' for their support throughout our educational career, especially in the last year our degree. They have always supported and encouraged to do our best in all matters of life.

We would also like to thank Mr. Zafar Gilani who was always there to support and guide us at every stage. He always motivated us. We also thank Mr. Faisal Zahid and Imran Ashraf as well to provide us all the prerequisites and technical support.

Finally, this project would not have been possible without the expert guidance of our advisor, Dr. Anjum Naveed, who has been a great source of inspiration for us during this project. We would also like to thank Dr. Zahid Anwar the co-Advisor of this project.

Despite all the assistance provided by Dr. Anjum Naveed and others, I alone remain responsible for any errors or omissions which may unwittingly remain.

Ghulam Nabi

Farhan Maqsood

Bilal Naseer

June, 2011

ABSTRACT

PingER is an internet performance monitoring tool being jointly managed by SEECS-NUST and SLAC, USA. Different PingER nodes worldwide are sent ping messages from each other and data is collected first at node level and then at a central location. Almost 1000 remote and monitoring nodes allow PingER to measure the performance of 99% of the world wide network deployment. Currently PingER infrastructure is based on flat-files. With the passage of time as data increases, difficulty in searching and management of data increases therefore there is a need to change this architecture from flat-files to Relational Database architecture so that it becomes easy to manage data and it also helps in calculation of results from data in tables instead of flat-files. First part of this project is to convert the entire data storage and analysis system architecture from flat files to Relational Database driven architecture. This should result in decrease in analysis time and improve scalability of PingER.

Second part of the project is to integrate the Constraint Based Geo location technique (CBG) with TULIP, a GEO location tool already deployed at SEECS and SLAC. The objective of the project is to use extensive infrastructure of PingER and multiple techniques to get more accurate location results.

Table of Contents

I. Part I: Analysis and Implementation of Relational Archive Site For PingER.....	13
1 INTRODUCTION AND BACKGROUND.....	14
1.1 PingER MECHANISM.....	14
1.2 PingER ADVANTAGES ^[2]	15
1.2.1 Comparisons with Economic and Development Indicators	15
1.2.2 Calculating International Bandwidth	16
1.3 PingER INFRASTRUCUTURE.....	16
1.3.1 Remote Sites:	16
1.3.2 Monitoring Sites:	16
1.3.3 Archive Sites:.....	17
1.4 MOTIVATION:.....	17
1.4.1 Reasons for Moving From Flat File to Database Approach ^[3]	19
2 LITRATURE REVIEW	20
2.1 ARCHITECTURE OF PingER:.....	20
2.2 MONITORING SITE FUNCTIONALITY:	21
2.3 ARCHIVE SITE FUNCTIONAALITY:	23
2.4 NODEDETAILS DATABASE ^[4]	23
3 METHODOLOGY	26
3.1 1 st LEVEL NORMALIZED SCHEMA:	26
3.1.1 Nodes Table	26
3.1.2 Ping_Data Table.....	27
3.1.3 Analysis Table	27
3.2 3 rd LEVEL NORMALIZED SCHEMA:.....	28

3.2.1	Nodes Table	29
3.2.2	End Points Table	29
3.2.3	Data Level 0 Table.....	29
3.2.4	Data Level 1 Table.....	29
3.2.5	Analysis Table	29
3.3	STRESS TESTING PROCEDURE:.....	31
3.4	STRESS TESTING RESULTS:	31
3.5	IMPLEMENTATION:.....	34
3.5.1	Flat-files Based Architecture:	35
3.5.2	Relational Database Architecture:	36
3.6	ADDING MOS AND ALPHA:	45
3.6.1	Mean Opinion Score (MOS):.....	46
3.6.2	ALPHA:	47
4	RESULTS	50
5	DISCUSSION AND RECOMMENDATION	55
5.1	DISSCUSSION.....	55
5.2	RECOMMENDATIONS:.....	55
6	CONCLUSION:.....	56
II.	Part II: CBG Integration with TULIP	57
7	INTRODUCTION AND BACKGROUND.....	58
7.1	GEOLOCATION ^[9]	58
7.2	IMPORTANCE OF GEOLOCATION	58
7.3	TYPES OF GEOLOCATION.....	59
7.3.1	CBG	59
7.3.2	GeoIP	59

7.3.3	Domain Name Services.....	60
7.3.4	Autonomous system.....	60
7.4	TULIP ^[7]	60
7.4.1	Reflector.cgi.....	60
7.4.2	Land marks.....	60
7.4.3	Sites.xml.....	61
7.5	MOTIVATION FOR INTEGRATION	62
7.6	PROBLEM STATEMENT	62
8	LITERATURE REVIEW	64
8.1	CONSTRAINT BASED GEOLOCATION.....	64
9	METHODOLOGY	69
9.1	ARCHITECTURE OF INTEGRATION	69
9.2	TOOLS USED	71
9.3	TULIP CONFIGURATION AT SEECS	71
9.3.1	Software and Services.....	72
3.1.1	Running Procedure.....	72
9.4	MATLAB SERVER CONFIGURATION AT SEECS	72
9.5	INTEGRATION METHODOLOGY WITH TULIP	73
9.6	MULTITHREADING.....	74
9.7	TULIP VISUALIZATION ^[11]	75
10	RESULTS AND PERFORMANCE EVALUATION	78
10.1	RESULTS	78
10.1.1	Comparison of Lat/Lon.....	78
10.1.2	Ideal Case.....	79
10.2	ACCURACY OF DIFFERENT CONTINENTS.....	80

10.2.1	Accuracy of Asia.....	80
10.2.2	Accuracy of Europe	82
10.2.3	Accuracy in Africa.....	84
10.2.4	Accuracy in North America	85
10.2.5	Average distance error comparison w.r.t continents.....	87
10.2.6	Average distance error comparison w.r.t countries.....	87
11	DISCUSSION	90
12	CONCLUSION.....	91
13	FUTURE WORK.....	93
13.1	USING BESTLINE APPROACH	93
13.2	LANDMARK TIERING ANALYSIS	94
14	REFERENCES	95
15	APPENDICES	96
15.1	CHANGES IN NODE.PL.....	96
15.2	CHANGE IN GETDATA.PL	96
15.3	CHANGES IN ANALYZE HOURLY	96
15.4	CHANGES IN ANALYZE-DAILY.PL	97
15.5	CHANGES IN ANALYZE-MONTHLY.PL.....	98
15.6	CHANGES IN PINGTABLE.PL.....	98
15.7	CHANGES IN CBG2.M.....	108
15.8	CHANGES IN CBG_SOI.M	108
15.9	CHANGES IN AUTOMATETEST.JAVA	108
15.10	CHANGES IN GetPingDataPL.java.....	110
15.11	SERVER_TULIP.PL	110
15.12	SERVER_CBG.PL	110

LIST OF FIGURES

Analysis and Implementation of Relational Archive Site for PingER

Figure 1: Ping Mechanism	15
Figure 2: PingER nodes worldwide	17
Figure 3: PingER Architecture	21
Figure 4: 1 st level normalized schema	28
Figure 5: 3 rd Level Normalized Schema ERD Diagram	30
Figure 6 : Flat Files Based Architecture	35
Figure 7: Relational Database Architecture	36
Figure 8: NODE DETAILS Table Data	37
Figure 9: Data Servers in NODEDETAILS	38
Figure 10: List of Monitoring Sites	38
Figure 11: Remote Sites	38
Figure 12: Metadata of Node_Details Table	39
Figure 13: Data Format in file	40
Figure 14: Metadata of pingdata	42
Figure 15: Metadata of Analysis Table	44
Figure 16: Result Output	51
Figure 17: Node_Details table	52
Figure 18: Pingdata Table	53
Figure 19: Analysis Table	54
CBG Integration with TULIP	
Figure 20: Sites.xml	61
Figure 21: CBG Mechanism	65
Figure 22: CBG Flow Diagram	68

Figure 23: TULIP CBG Integration	71
Figure 24: TULIP Interface before CBG Integration	76
Figure 25: TULIP Interface after Integrating CBG	77
Figure 26: Results Table	78
Figure 27: Without Landmarks	79
Figure 28: Accuracy in Asia graph	82
Figure 29: Accuracy in Europe graph	83
Figure 30: Accuracy in Africa graph	85
Figure 31: Accuracy in North America	86
Figure 32: Average distance error w.r.t to continents	87
Figure 33: Average distance comparison world wide	89

List of Tables

Table 1: Node Details table.....	23
Table 2: 1 st level Normalized Schema Stress Testing.....	31
Table 3: 3 rd Level Normalized Schema Stress Testing Results	32
Table 4: Flat Files Stress Testing Results	33
Table 5: Ideal Case.....	80
Table 6: Accuracy in Asia.....	80
Table 7: Accuracy in Europe	82
Table 8: Accuracy in Africa.....	84
Table 9: Accuracy in North America.....	85
Table 10: Average distance error comparison w.r.t to world wide.....	87

List of Abbreviations

PingER	Ping End-to-end Reporting
IEPM	Internet End to end performance Monitoring
RTT	Round Trip Time
ITU	International Telecommunication Union
UN	United Nation
PERN	Pakistan Educational and Research Network
HTTP	Hypertext Transfer Protocol
NOMN	Number of Monitoring Nodes in nodes.cf
NOMT	Number of Matrices
IQR	
ERD	Entity Relationship Diagram
MOS	Mean Opinion Score
PerfSonar	PERFormance Service Oriented Network monitoring ARchitecture.
SQL	Structure Query Language
RTD	Round Trip Distance
GPS	Global Positioning Systems
SLAC	Stanford Linear accelerator center
TULIP	Trilateration Utility for Locating IP hosts
CBG	Constraint Based Geolocation
VPNS	Virtual Private Network Systems
DNS	Domain Name Services
AS	Autonomous Systems

I. Part I

**Analysis and Implementation
Of
Relational Archive Site
For
PingER**

1 INTRODUCTION AND BACKGROUND

PingER is an Internet Performance Measurement utility to monitor end-to-end performance of Internet links, developed by the IEPM group at the Stanford Linear Accelerator Center (SLAC). The network performance of more than 950 hosts is monitored worldwide. This covers 99% of the internet population of the world. PingER provides insight into a multitude of network activity. To summarize the results, it is necessary to aggregate the measurements by regions and to divide them into measurements of long-term trends and of short-term glitches. PingER also helps in providing the better expectation of network performance between the sites that are monitored worldwide.

1.1 PingER MECHANISM

PingER uses the ubiquitous ping facility that works on Internet Control Message Protocol (ICMP) echo mechanism. Ping facility allows you to send a packet of a user selected length to a remote node and have it echoed back. Nowadays it is available as a pre-installed facility on all platforms. So it can be used directly by the pingER application that runs on high priority (usually on kernel of Linux). Running PingER as root process helps in measuring performance more accurately. Another advantage of using ping is that it provides low intrusiveness (~100 bits per second per monitoring-remote-host-pair).^[1]

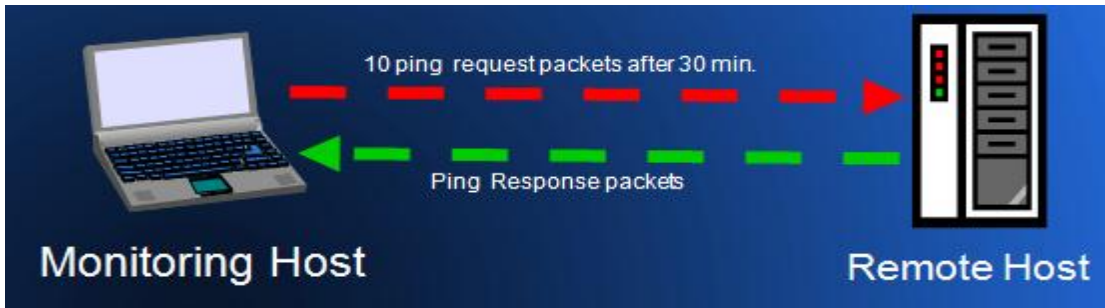


Figure 1: Ping Mechanism

1.2 PingER ADVANTAGES ^[2]

PingER calculates different matrices e.g. RTT, packet loss, jitter, IQR, ping unreachability, out of order packets etc. to evaluate the performance of the network links between the nodes which are being monitored. The calculated and aggregated values of these matrices help in identifying those sites which have bad network performance. For example if packet loss is 4-6% then video conferencing becomes irritating, occurrence of long delays of 4 seconds or more at a frequency of 4-5% or more is also irritating for interactive activities such as telnet and X windows. We can use this data to identify sites to be upgraded, identify last miles problems, setting expectations for collaborations where large data is transferred. This can also be used for choosing routes with small RTT and minimum packet loss, setting expectations for VoIP and quantifying the digital divide.

1.2.1 Comparisons with Economic and Development Indicators

The various organizations like ITU, UN and World Bank have setup different indices e.g. life expectancy, GDP, literacy, phone lines, Internet penetration etc. to measure the development in countries. Many of these matrices require a lot of time and cost for measurement. So they have become outdated in some cases. One of the most important factors determining the economic development of a country in today's information age is its Internet connectivity. The economic development and internet connectivity are strongly correlated to each other. PingER's Internet measurements

can help in characterizing a country's development is due to the fact that PingER's data is current (up-to date) and covers most of the countries of the world.

1.2.2 Calculating International Bandwidth

As PingER's infrastructure covers huge part of internet's population, the matrices calculated by PingER help in calculating the international bandwidth capacity according to some w.r.t countries. Now let's see some interesting case studies related to the use of PingER.

1.3 PingER INFRASTRUCUTURE

PingER has a large infrastructure working worldwide. Whole functionality is controlled with the help of different types of nodes that exist in different regions of world.

There are three types of nodes in PingER architecture.

1.3.1 Remote Sites:

These are simple passive nodes that are discoverable and can reply to a ping via network. There are approximately 900 remote sites so far, that exist in different regions of the world.

1.3.2 Monitoring Sites:

Monitoring Sites ping the remote and other monitoring sites to measure RTTs. These nodes have been installed with PingER monitoring tools. The data collected by monitoring sites is stored and is also made available to archive site for analysis. Some of these monitoring sites that show regular and consistent online behavior and also provide complete data for analysis are called Beacon sites. As of May 2011 there are 87 monitoring (including Beacons) sites are active and this number is increasing every year.

1.3.3 Archive Sites:

The archive sites gather the information, by using HTTP, from the monitor sites at regular intervals and archive it. In PingER architecture there should be at least one Archive site. PingER has currently more than one working archive sites. Archive site provide the archived data to the analysis site(s) that in turn provide reports that are available to users via the Web. Archive site and Analysis site can be on the same machine or server.

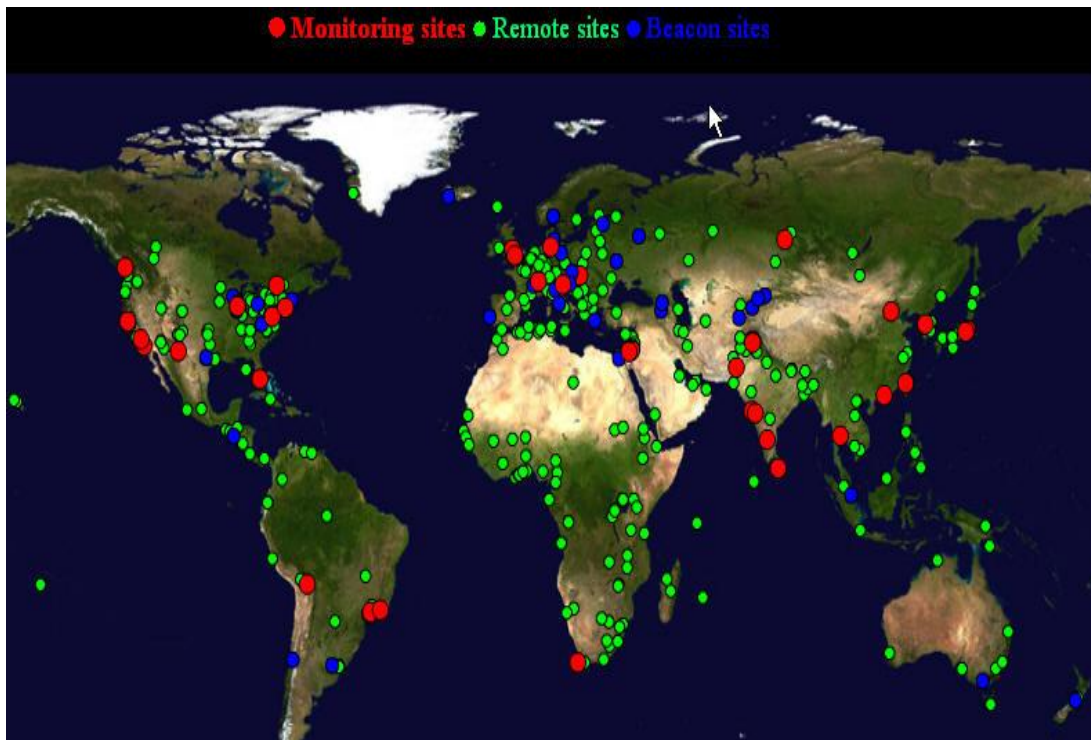


Figure 2: PingER nodes worldwide

1.4 MOTIVATION:

PingER archive site was first constructed a decade ago. It was based upon flat files to store archive data and analysis reports. Using flat file system was feasible at that time because data was not available in huge amount. But with the passage of time the internet usage increased worldwide (number of hosts' increased), as a result of which monitoring infrastructures of various links also became complex. This increase in number of monitoring hosts increased the data for analysis hugely by the addition of

hundreds of files with thousands of records in each file every day. This exponential increase of data resulted in increase of processing time for analysis operations. Because the raw data available in flat files collected from monitoring nodes is non-relational or unstructured. Therefore, to make the analysis of data robust for current amount of data there is a need to shift PingER archive site architecture from flat files to relational database. So the purpose of this project is to implement a relational archive site which stores raw data in tables instead of flat files and operations of analysis i.e. aggregation or summarization of data are also performed on these data tables by executing SQL queries instead of parsing flat files. This will result in huge reduction of analysis time. After this the analyzed data will also be stored in tables instead of flat files for display or measuring network performance of internet links. This whole process will be transparent to end users because the whole interface will remain the same.

Secondly the matrices that PingER calculates are

1. Conditional loss probability
2. Ping unreachability
3. Ping unpredictability
4. Round Trip Time
5. Packet Loss
6. Inter Packet delay variation or Jitter
7. TCP throughput
8. Out of order Packets
9. Duplicate Packets
10. Inter Quartile Range
11. Zero Packet loss frequency

There are two other matrices Alpha and Mean Opinion Score (MOS) which are being widely used. Telecom Companies use MOS as a metric of voice quality. And alpha is used to measure the directness of routes. To increase the efficiency and usability these

two matrices should also be added in the current version of PingER which is working on flat files as well as new relational database architecture.

1.4.1 **Reasons for Moving From Flat File to Database Approach** ^[3]

1. Database provides a relational data approach. This helps in easy retrieval of data.
2. A good database design will decrease redundancy of information as compared to flat files. Where flat files would have the similar size for each new file, a database table will not grow in the same way since repetitive fields are already normalized.
3. Relational database has quicker retrieval time as compared to flat files when data-analysis tools are taken into account. Where data has to be read and split atomically into correct fields, a database table already provides information atomically.
4. Database approach in the system will provide better scalability
5. The invalid and null values in the data can also be replaced by using relational approach relatively easy as compared to flat files.

2 LITRATURE REVIEW

To better understanding the functionality of PingER, first of all we will explain the data structures at different types of nodes and their functionalities.

2.1 ARCHITECTURE OF PingER:

The performance of the network is determined by analyzing data which is collected by continuous ping of monitoring nodes to remote nodes. This data is then collected from monitoring nodes by an **Archive Site** on which analysis of data takes places. Originally all the collected data which is available at archive site is in flat files and all the analysis operations are performed by fetching appropriate data from these flat files. The result of this analysis is also stored in flat files which are later used for display.

The basic concept is that all the monitoring sites ping remote sites and other monitoring sites and collect data i.e. RTTs of packets. Then Archive site collects and archives all the data which is collected by these monitoring sites on daily basis. After this various scripts perform analysis on this archived data and store results for determining network performance of different links. Analysis is performed according to different metrics for different regions given different periods of time etc. The architecture diagram of PingER is shown below which shows all node types and data flow at them.

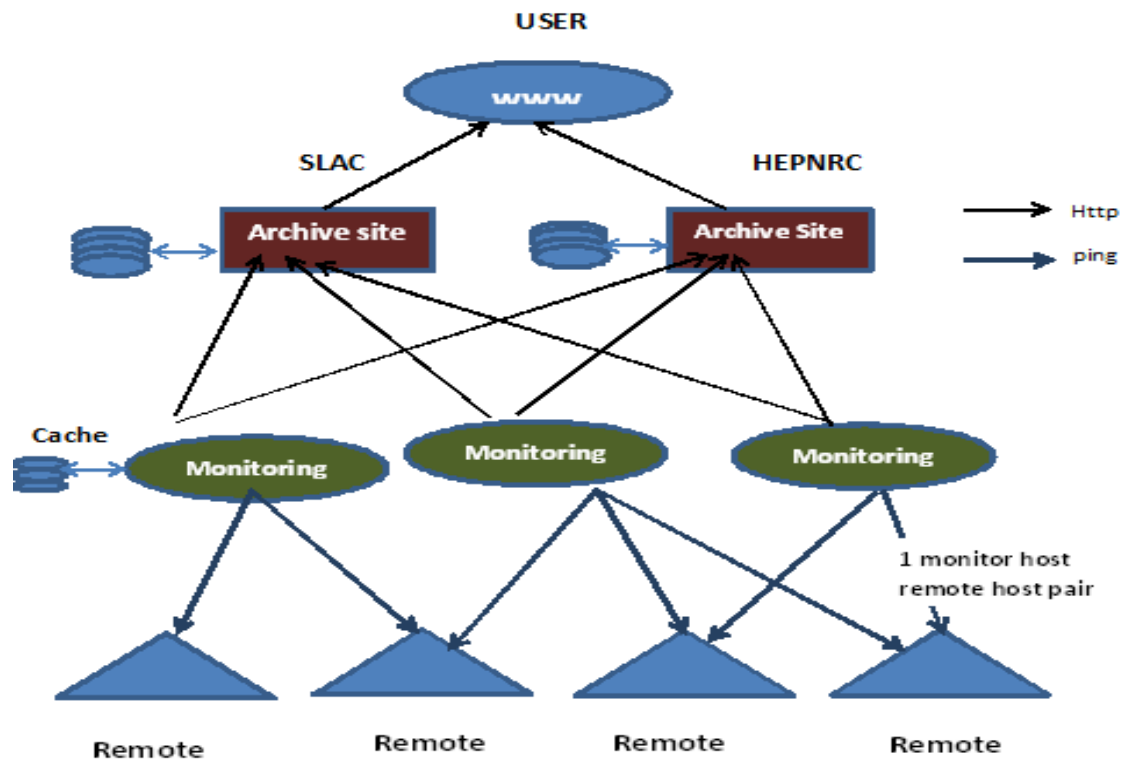


Figure 3: PingER Architecture

Before we start discussion on archive site it is necessary to understand the procedures that take place in monitoring site so that we can understand basics of data structures of an archive site. To make a host to act as a monitoring node we install PingER software on it.

2.2 MONITORING SITE FUNCTIONALITY:

All monitoring nodes have **pinger2.pl** software installed on them. This script runs after every thirty minutes as cronjob and generates 21 pings every time to all remote host listed in a file named **Beacons.txt**. As the script runs and pings remote hosts, data i.e. RTTs and number of packets sent and received are collected and stored in a file named as **ping-yyyy-mm.txt** in a directory at each monitoring site. So for one complete month the data collected by a monitoring node by pinging remote hosts after every thirty minutes is stored in this file with particular month included in the file name.

The format of one line or record of this file is shown below:

```
Monitoring node name      Monitoring node IP  Remote node name  Remote
node IP Packet size  UNIX epoch time    Packets sent  Packets      received
      Minimum RTT Average RTTMaximum RTT      Sequence numbers (1-10)
      RTTs (1-10)
```

For example:

```
pinger.slac.stanford.edu    134.79.240.30      ping.slac.stanford.edu
134.79.18.21 100
1152489601  10  10  0.255  0.341  0.467
0    1    2    3    4    5    6    7    8    9
0.287  0.380 0.467 0.391  0.327  0.387  0.291  0.332  0.255  0.299
```

Similarly:

```
monitor.niit.edu.pk 203.99.50.204 www.carnet.hr 161.53.160.25 100 1171756802 10
10 223.323 224.978 226.805 1 2 3 4 5 6 7 8 9 10 226 223 226 223 223 226 223 224
226 225
```

The number of lines in each file is calculated approximately by following formula:

$$\text{Number of Lines} = \text{Beacons} * 48 * 30$$

Where Beacons are the remote hosts listed in file beacons.txt. A script named pinger2.pl at monitoring sites runs after every thirty minutes to ping all remote hosts so for one day or 24 hours it runs $24*2=48$ times and there are 30 days in a month. So if number of beacons or remote hosts in beacons.txt is 300 then number of lines in this data file of each monitoring node is $300*48*30=432000$ lines or records every month on each monitoring site. As the number of monitoring and remote hosts is increasing, the number of lines or records is also increasing. This results in increase in processing time for analysis of this huge unstructured and non-relational data at archive site where all of this data from all monitoring nodes is collected.

2.3 ARCHIVE SITE FUNCTIONAALITY:

At Archive site a script named `node.pl` runs as a cronjob four times a day to collect updated information of nodes from PingER (NODEDETAILS) database, which is placed at SLAC. This nodes information is stored in a file named `nodes.cf` at Archive site. All the data collecting scripts i.e. `getdata.pl` or analysis scripts i.e. `analyze-hourly.pl`, `analyze-daily.pl` etc. use this information stored in `nodes.cf` file when dealing with different types of nodes. This script is also used for updating node information, adding a node, deleting a node, find the region, continent or IP address or related information of a node or filtering a particular type of nodes etc.

2.4 NODEDETAILS DATABASE ^[4]

The metadata of **NODEDETAILS** table whose data is collected into `nodes.cf` file is shown here:

Table 1: Node Details table

Name	Null?	Data Type	Use
NODENAME	NOT NULL	VARCHAR2(100)	DNS host name
IPADDRESS		VARCHAR2(15)	IPv4 address
SITENAME		VARCHAR2(100)	Domain name of the node
NICKNAME		VARCHAR2(35)	Abstraction of the hostname with the TLD first ¹
FULLNAME		VARCHAR2(100)	Human-friendly description of the node/site
LOCATION		VARCHAR2(100)	City and/or State/Province/Region for node ²
COUNTRY		VARCHAR2(100)	Country for node
CONTINENT		VARCHAR2(100)	Continent or region where node is thought to be located ³
LATANDLONG		VARCHAR2(25)	Latitude and longitude of node
PROJECTTYPE		VARCHAR2(10)	Flags describing how nodes are used ⁴
PINGSERVER		VARCHAR2(100)	URL for requesting a ping from this node to another ⁵

TRACESERVER		VARCHAR2(100)	URL for requesting a traceroute from this node to another ⁵
DATASERVER		VARCHAR2(100)	URL for retrieving PingER data from this node ⁵
URL		VARCHAR2(100)	URL for the home page for the institution running the node
GMT		VARCHAR2(10)	Node's time offset from GMT, not used
COMMENTS		VARCHAR2(4000)	Comments and notes on when and how the node's record was last updated ⁶
APP_USER		VARCHAR2(20)	Windows user name of the last user to edit the node's record through the UI
CONTACTS		VARCHAR2(100)	Name and email address(es) of the node's maintainer(s)
PING_SIZE		NUMBER	Size of pings to be sent to the node - only controls SLAC's PingER install

On archive site the script **getdata.pl** collects data from monitoring sites and stores in a file named like **ping-yyyy-mm-dd.txt.gz**. This script runs on daily basis and collects data for particular date in a file which is included in the file name. The format of this file is same as of file in monitoring sites described above. This script reads all monitoring nodes which are listed in **nodes.cf** file and collects raw or ping data from these monitoring sites and stores this data at a directory in archive site.

On the other hand at archive site there are analysis scripts i.e. **analyze-hourly.pl**, **analyze-daily.pl**, **analyze-monthly.pl**, **analyze-yearly.pl**, **analyze-allyears.pl** etc. These scripts use the data that is stored at a directory by **getdata.pl**. These scripts use different mathematical analysis operations like summarization or aggregation(some other functions also) for generating results about **packet loss, IQR, Jitter and Throughput etc.** of different links in different regions with different packet size for different periods of time. After analyzing records the results are stored in files having names of the form **packet_loss-1000-by-node-2011-02-11.txt.gz**, **average_rtt-100-by-site.txt.gz**, and **throughput-100-by-node-days-120.txt.gz** and **iqr-1000-by-site-2011-02.txt.gz** at archive site.

Previously we approximately calculated the number of lines or records in a monthly data collecting file at monitoring node. As in archive site the daily data from these files are archived, analyzed and also stored in another type of files so there is a need to know how many files the archive site may generate daily to analyze the complexity of this flat files based system. So the number of files of collected and analyzed data created in each month is calculated as:

$$\text{Number of files/month} = (\text{NOMN} + \text{NOMT} * 5) * 30$$

Here

MOMN: Number of monitoring nodes listed in **nodes.cf** file (Every day one file per monitoring host is generated).

NOMT: Number of metrics i.e. **packet-loss** or **Jitter** (As daily for each metric 3 analysis scripts generate 5 different files).

So if number of monitoring nodes is 75 and number of metrics are 12 then number of files generated each month is $(75+12*5)*30=4050$ and number of records or lines in each file is calculated by **Beacons * 48 * 30** as described above.

So this exponential increase in number of records and files leads us to a relational way to store and analyze by shifting this whole files based architecture to database architecture so that all the data is stored in database instead of files. We should change different scripts accordingly such that data is collected in to tables and after analysis results are also stored in tables without changing the interface.

3 METHODOLOGY

We intend to convert the PingER archive site architecture to a more robust, efficient and modern one. For this purpose we introduced two schemas of implementation and performed stress testing on each schema. We also conducted a stress test for flat files. Our stress testing involves time and space complexity for finalizing which option performs optimally over a set of constraints.

3.1 1st LEVEL NORMALIZED SCHEMA:

It is level-1 normalized and is relatively simple. Time of insertion and retrieval of data is less as no joins are required for it. Its design is simply based on data structures of ping data files i.e. “**ping-2010-02-04.txt.gz**” and the files which stores analysis results i.e. “**packet-loss-100-by-site-2010-12-11.txt.gz**”etc.

In this schema there are three tables

3.1.1 Nodes Table

Nodes table is basically the modified form of NODEDETAILS table which is placed at SLAC in old PingER architecture and is described above. This table contains all the information about the nodes in its fields like node names, IP addresses, lat / long, country, region, ping_server, trace route server, Data server, project_type etc. The information stored in this table is used for gathering and analysis of data different nodes. The primary key of this table is Host_id. Project_type is a flag which tells what type of functionality a node has e.g. whether it remote, monitoring or a beacon site. Ping_server, traceroute_server and data servers are URL’s of different servers that are used in either gathering or analysis process.

3.1.2 Ping_Data Table

Ping_Data table is used to store the data which is gathered at the monitoring sites (by pinging remote sites and other monitoring sites). This table has monitoring host id, remote host id, UNIX epoch time, packet size, number of packets sent and received and sequence number and RTT's of each packet. Monitoring host id and remote host id are basically the foreign keys from Nodes table. From number of packets sent and received, we can directly calculate packet loss. While from RTT's we can find out minimum, maximum and average RTT's directly or simple aggregation operations.

3.1.3 Analysis Table

After the data stored in Ping_Data table is analyzed by analysis scripts, the results are stored in Analysis table. Analysis table has a composite primary key which contains six fields to uniquely identify each row. Each row contains all the metrics of analysis for each unit of time i.e. a row contains analysis data for each hour which is used for analyzing data on daily basis and this daily analysis data is used for calculating monthly data.

This table is constructed in such a way that analysis data of each metric e.g. Packet loss, ping unreachability, minimum RTT, duplicate packets and IQR etc. according to each time interval like hourly, daily, monthly, yearly etc. can be retrieved directly by simple queries. This is the data that can be used to predict performance of some internet link.

Below is the ERD diagram of 1st level normalized schema.

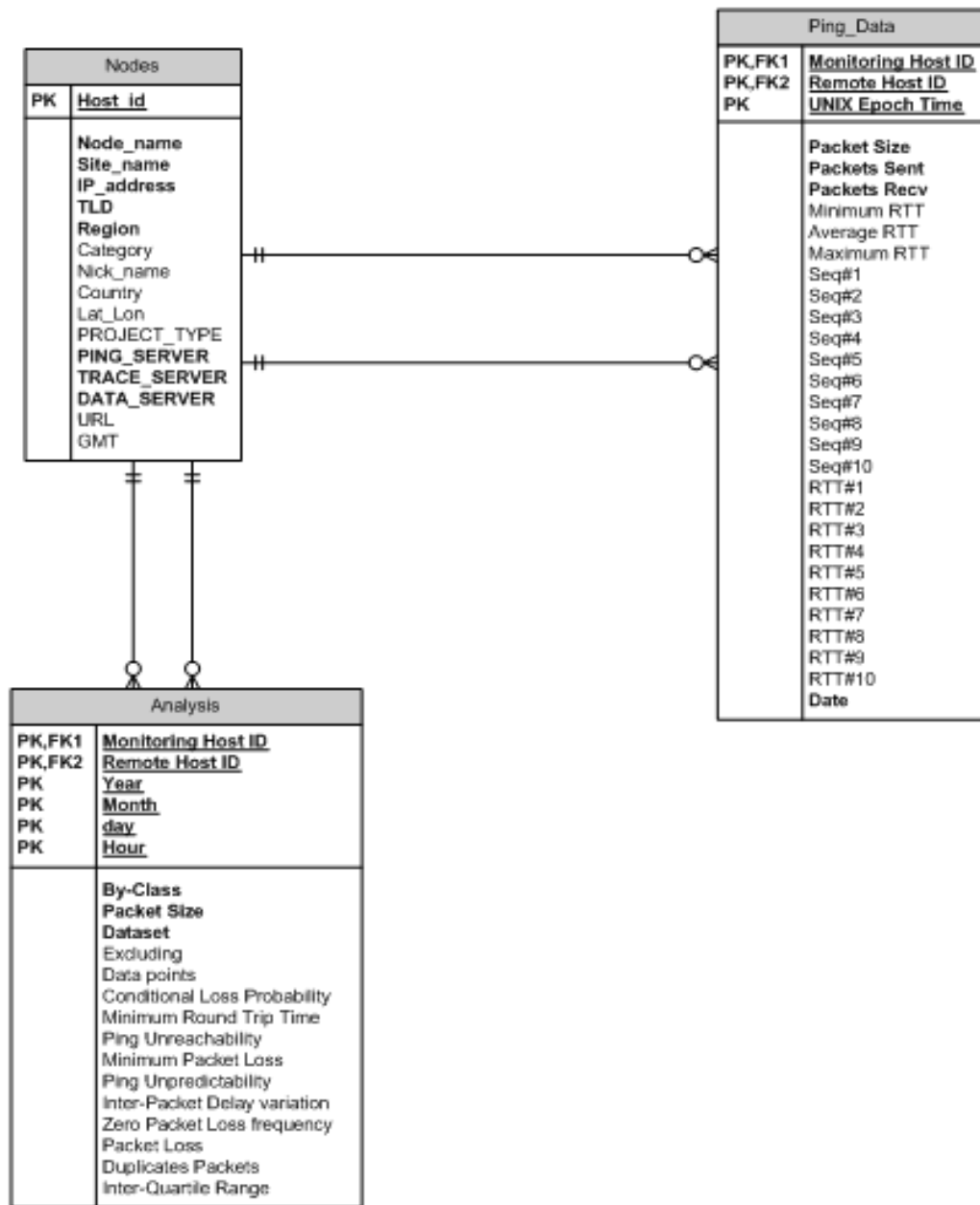


Figure 4: 1st level normalized schema

3.2 3rd LEVEL NORMALIZED SCHEMA:

This is level-3 normalized and is relatively complex. Time of insertion and retrieval of data is comparatively higher because joins are required for retrieving required data from files. Design is also based on data structures of ping data files and the files which stores analysis results but in a normalized manner.

There are 5 tables in this schema.

3.2.1 **Nodes Table**

The Nodes table in this schema is same as described above in 1st level normalized Schema and is modified form of NodeDetails table which is currently on SLAC.

3.2.2 **End Points Table**

This table contains all monitoring host remote host pairs. It has its own primary key whereas the monitoring host id and remote host id are foreign keys from Nodes table. This means we can access a pair with single key instead of accessing two both id's separately. Packet size is also included in this table because it remains same for one monitoring remote pair usually. This table will only be updated when some new node is inserted in the nodes table.

3.2.3 **Data Level 0 Table**

This table contains the number of packets sent, received, Date, UNIX epoch time, minimum, maximum and average RTT, and end pint id which is the foreign key from End Points table.

3.2.4 **Data Level 1 Table**

This table only stores the sequence numbers of packets and their respective RTT's according to each row of Data Level 0 table.

3.2.5 **Analysis Table**

Analysis table in 3rd level normalized is identical to the one in the 1st level normalized Schema because the final output and calculated matrices remain same.

Here is the ERD diagram of 3rd level normalized Schema

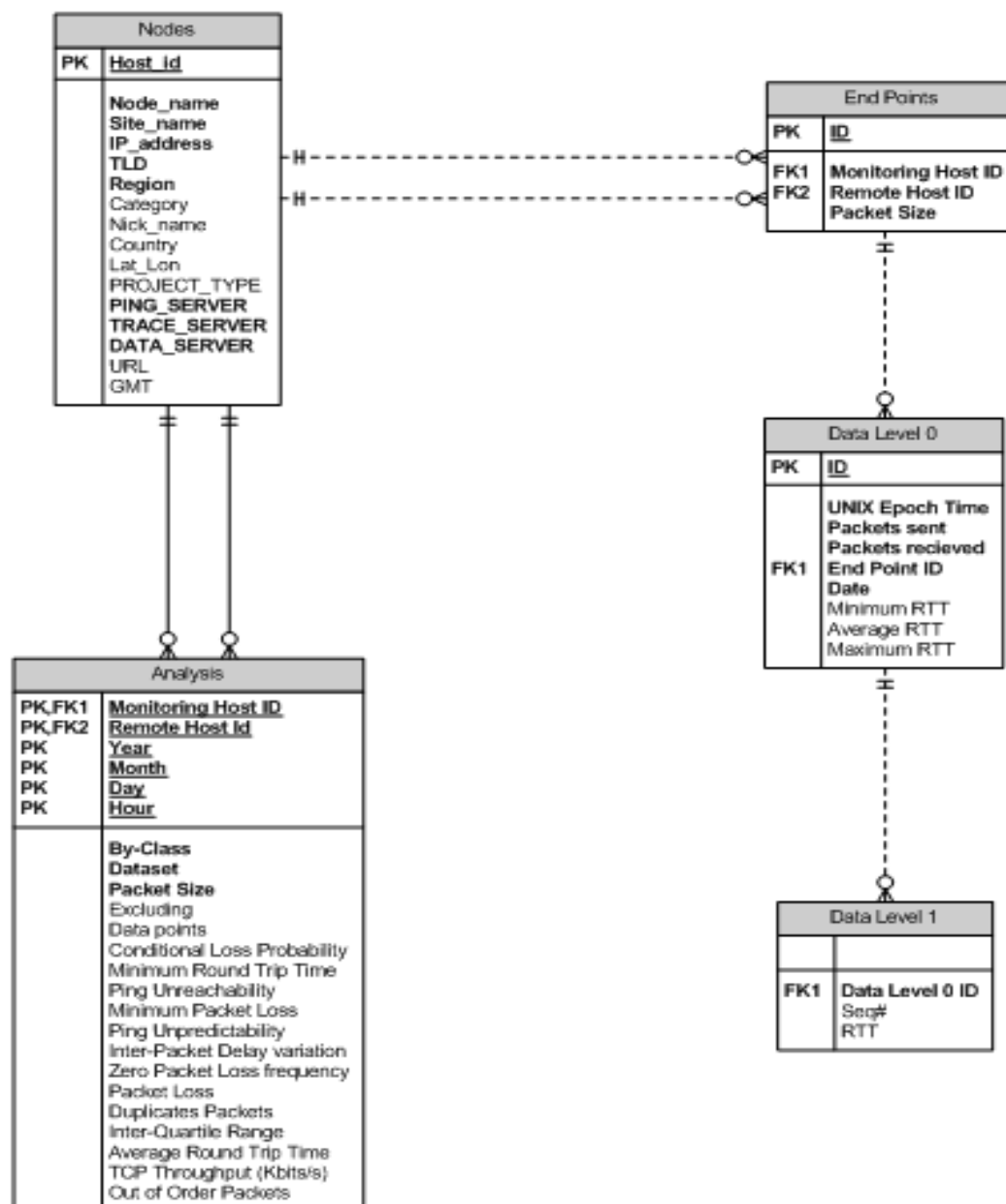


Figure 5: 3rd Level Normalized Schema ERD Diagram

In the above schemas the Nodes table is the modified form of NODEDETAILS table. By modifying this table, **nodes.cf** is not used anymore. The data contained by this file is now directly accessed from **nodes** table. **Ping_data** table have same fields as of one line of file named as **ping-yyyy-mm-dd.txt.gz** file described above. Due to

which this file is also removed. Both of these tables relate to each other by **Host_id**. Analysis table has a structure that is able to store data (analysis results) of any analysis file from a number of different types of analysis files as described above in **Archive Site** section and removes the need for these separate analysis files.

3.3 STRESS TESTING PROCEDURE:

We measure Time and Space complexity of both of these schemas as well as flat files by creating two databases (according to both schemas). Time complexity is measured as insertion and retrieval time of data excluding time required for analysis operations on raw data (raw data means data in files like “**ping-2010-02-04.txt.gz**” i.e. RTTs collected by the **getdata.pl** script).Space complexity measures the space occupied by records on storage medium (in this case a hard disk drive). Space is measured in multiples of bytes. We are measuring it in Mega-bytes (MB) to keep it in human readable form. Space occupied is basically the SQL file size in case of databases and sum of file sizes in case of flat files.

The stress testing results and analysis of both schemas are shown below in order to choose one appropriate schema.

3.4 STRESS TESTING RESULTS:

Table 2: 1st level Normalized Schema Stress Testing

1stLevel Normalized Schema Stress Testing Results			
Number of files	Insertion Time complexity	Retrieval Time complexity	Space complexity
25	47.62s	22.56s	45.5 MB
50	1m32s	56.23s	92 MB
100	2m58s	1m48s	187.4 MB

500	13m34s	7m27s	970.7 MB
1000	28m12s	18m35s	1.8 GB

Here Insertion time complexity means the time to insert data into tables i.e. loading of data from files to tables. It does not include time taken by tables for analysis operations on data in it which should be much less as compared to Flat-files. Simply Retrieval time complexity means only the time taken by analysis operations for retrieving data from tables excluding any processing time on this data. Space complexity is the size of an SQL file containing records of a particular number of files e.g. Loading data of five files into tables and calculate size of its SQL file etc.

Table 3: 3rd Level Normalized Schema Stress Testing Results

3rd Level Normalized Schema Stress Testing Results			
Number of files	Insertion Time complexity	Retrieval Time complexity	Space complexity
25	18m57.29s	24.25s	51.2 MB
50	41m9.32s	1m15s	99.6 MB
100	1h23m21s	4m13s	210.0 MB
500	7h46m13s	26m55s	1.24 GB
1000	16h25m44s	1h14m9s	2.56 GB

In Flat-files Insertion time complexity is time taken for loading data into files at archive site from files at monitoring sites. Similarly Retrieving time complexity is the time taken by scripts to retrieve data from files and Space complexity is the sum of sizes of individual files.

Table 4: Flat Files Stress Testing Results

Flat Files Stress Testing Results			
Number of files	Insertion Time complexity	Retrieval Time complexity	Space complexity
25	30.85s	17.78s	7 MB
50	55.451s	48.14s	14.5 MB
100	2m48s	1m49s	30.1 MB
500	16m47s	13m28s	157.4 MB
1000	43m31s	30m21s	321.5 MB

In **3rd level normalized** we basically divide **Ping_data** table in **1st level normalized Schema** into three tables **Endpoints, datalevel0 and datalevel1** by normalization.

By observing both schemas and their stress testing with respect to Flat-files we conclude that **1st LEVEL NORMALIZED-SCHEMA** should be our first choice replacement because **1ST LEVEL NORMALIZED-SCHEMA** performs better than **COMPLEX-SCHEMA**. It is both space and time efficient if compared to **COMPLEX-SCHEMA** and as shown in **Table-1** and **Table-2**. Though flat-files have less insertion and deletion times as compared to both of the schemas, it is however a slower and cumbersome approach when data analysis is taken into account. A select query can retrieve data atomically from one or more tables using a single statement. A file however needs to be read and processed to get data in atomic form.

3.5 IMPLEMENTATION:

For the implementation of this proposed architecture of PingER we modified scripts to store data into a database instead of flat files and also read values from database for operations. Originally node.pl script reads information about nodes from the NODEDETAILS table and stores in nodes.cf file. We changed **node.pl** script which now stores data in **Nodes** table instead of **nodes.cf** file (shown in 15.1). Therefore this file has been removed from PingER architecture and is replaced by Nodes table. We also changed **getdata.pl** script so that it can now store data into **Ping_data** table instead of files named as **ping-yyyy-mm-dd.txt.gz**. It is also changed to read monitoring nodes details from Nodes table instead of reading from nodes.cf file. (15.2)

Below are the architecture diagrams of flat filing based system and relational database based system.

3.5.1 Flat-files Based Architecture:

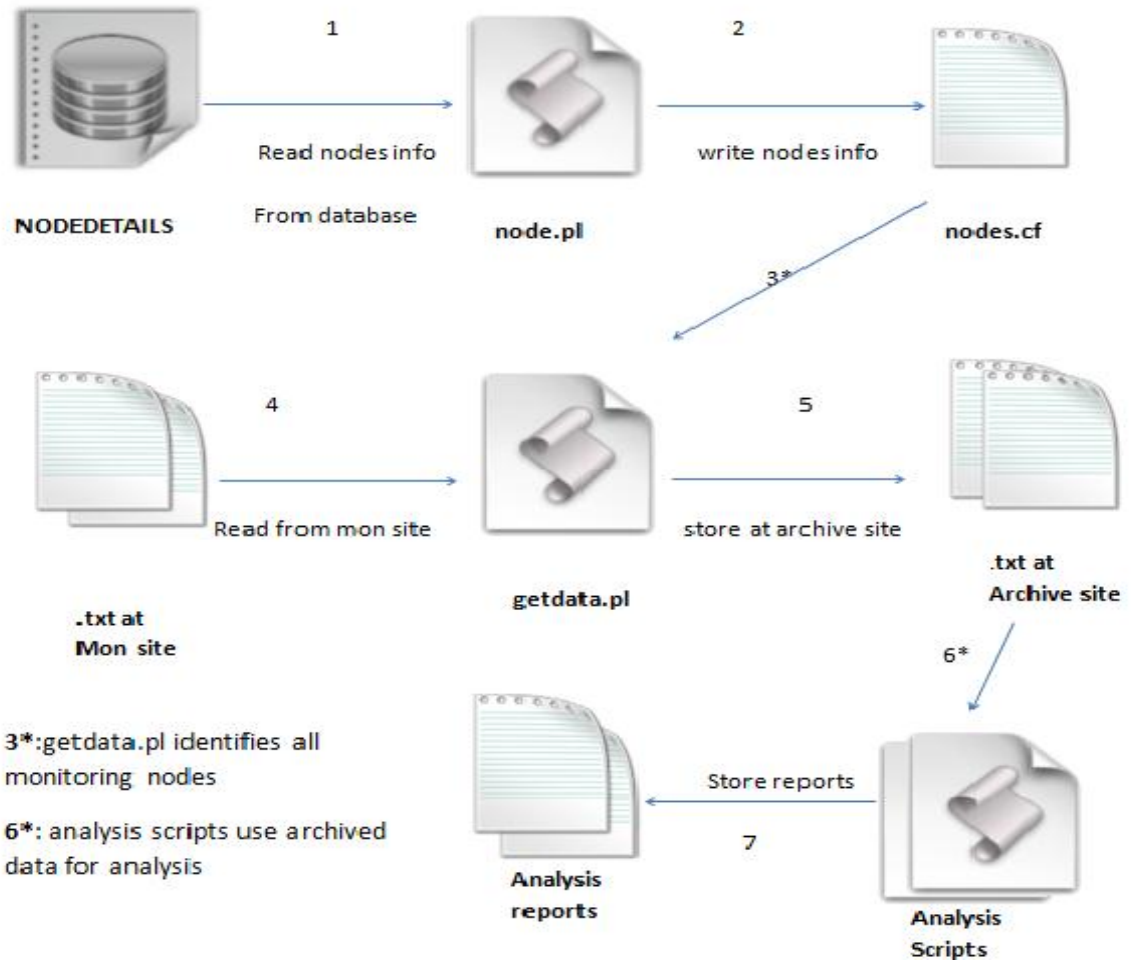


Figure 6 : Flat Files Based Architecture

3.5.2 Relational Database Architecture:

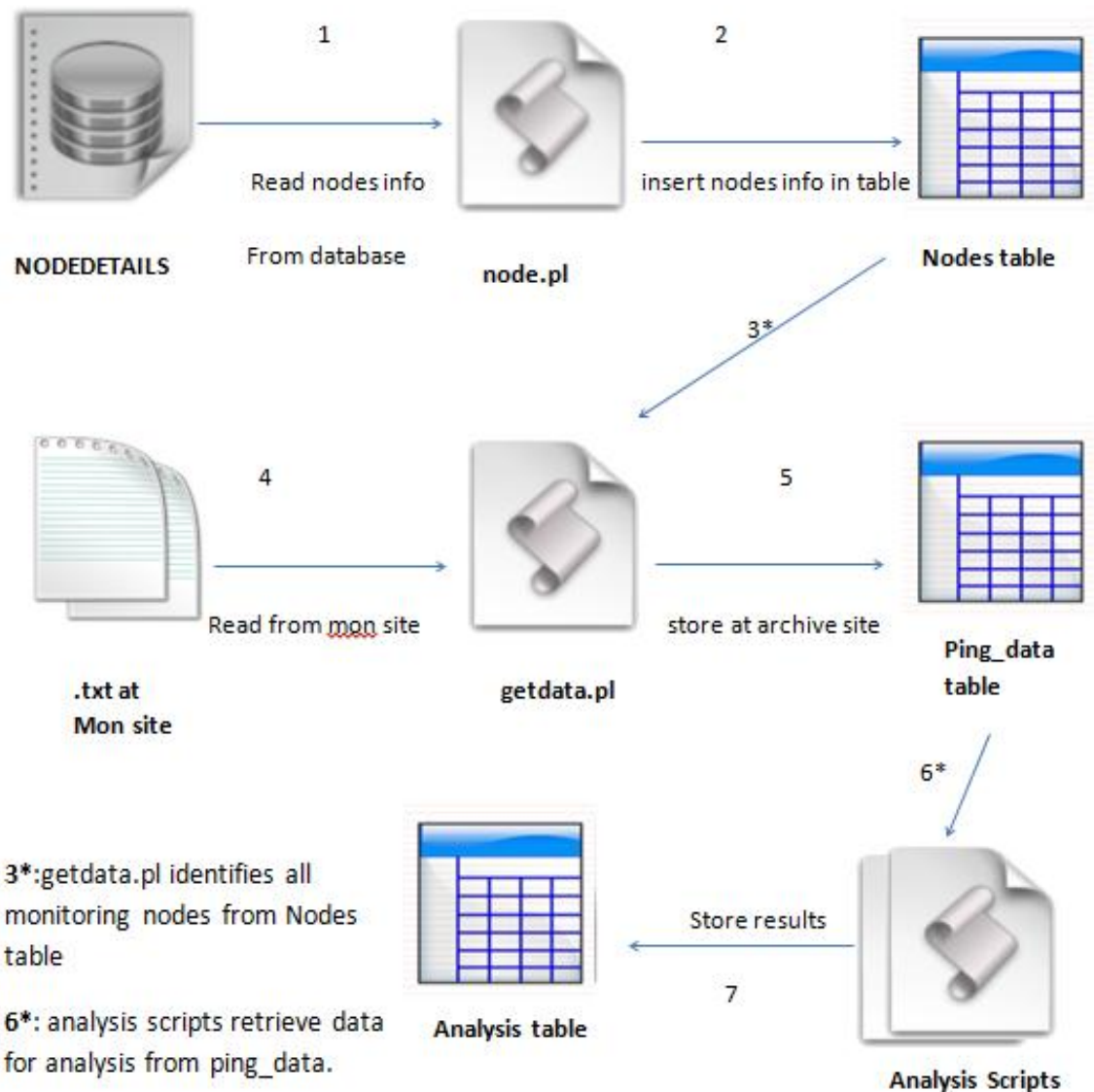


Figure 7: Relational Database Architecture

In the flat-files system all the information about nodes is placed in a file **nodes.cf**. It is a large file generated by script named **node.pl** which is running four times a day to keep nodes information up to date. The format of **nodes.cf** file is explained below:

The information of all nodes is placed in a **%NODE_DETAILS** key. The data for one node in this key is displayed in following figure:

```
"pinger.ascr.doe.gov" => [  
  "192.73.213.124",  
  "ascr.doe.gov",  
  "GOV.DOE.N8",  
  "US Department of Energy HQ",  
  "Germantown, Maryland",  
  "United States",  
  "North America",  
  "39.17 -77.25",  
  "M",  
  "http://pinger.ascr.doe.gov/cgi-bin/traceroute.pl?function=ping",  
  "http://pinger.ascr.doe.gov/cgi-bin/traceroute.pl?",  
  "http://pinger.ascr.doe.gov/cgi-bin/ping_data.pl?",  
  "http://www.er.doe.gov/ascr/",  
  "N/A",  
  " ",  
  " ",  
  ],
```

Figure 8: NODE DETAILS Table Data

All the URLs of monitoring site from which archive site collects raw data are placed in a separate key **%data_servers**. An example figure of this key is shown below:

```
%data_servers=(
  "49.50.236.98" => ["http://49.50.236.98/cgi-bin/ping_data.pl?", ],
  "airuniversity.seecs.edu.pk" => ["http://airuniversity.seecs.edu.pk/cgi-bin/ping_data.pl?", ],
  "brunsvigia.tenet.ac.za" => ["http://brunsvigia.tenet.ac.za/cgi-bin/ping_data.pl?", ],
  "cc.if.ufrj.br" => ["http://cc.if.ufrj.br/cgi-bin/ping_data.pl?", ],
  "cithep130.ultralight.org" => ["http://cithep130.ultralight.org/cgi-bin/ping_data.pl?", ],
  "davinci.ampath.net" => ["http://davinci.ampath.net/cgi-bin/ping_data.pl?", ],
  "falcon.cse.unsw.edu.au" => ["http://falcon.cse.unsw.edu.au/cgi-bin/ping_data.pl?", ],
```

Figure 9: Data Servers in NODEDETAILS

Similarly the monitoring and remote sites and countries and groups are stored in special arrays in this file. An example of such arrays is shown below:

```
@hep_monitoring_sites=( "AU.EDU.UNSW.N1", "BF.UNIV-OUAGA.N1", "E
D.UERJ.N2", "BR.ORG.SPRACE.N1", "BR.ORG.SPRACE.N2", "BR.UFRJ.N1"
.AC.N3", "DE.DESY.N1", "DZ.ARN.N3", "EDU.SDSC.N1", "EDU.SLAC.STAN
OE.N8", "GOV.ESNET.NOC.N4", "GOV.FNAL.N10", "HK.CSE.UST.N1", "IN.
NL.MUMBAI.N1", "IT.ICTP.N2", "JO.SESAME.ORG.N1", "JP.RIKEN.GO.N2
"MX.CUDI.JUAREZ.N1", "MX.NOC.CUDI.EDU", "MY.UNIMAS.N3", "NET.AMF
```

Figure 10: List of Monitoring Sites

```
@remote_countries=( "Afghanistan", "Albania", "Algeria", "Andorra",
an", "Bahamas", "Bahrain", "Bangladesh", "Belarus", "Belgium", "Benin
aso", "Burundi", "Cambodia", "Cameroon", "Canada", "Cape Verde", "Cha
yprus", "Czech Republic", "Democratic Republic of Congo", "Denmark
or", "Eritrea", "Estonia", "Ethiopia", "Faroe Islands", "Finland", "F
```

Figure 11: Remote Sites

There is a requirement to store all of this data in to a table in the same way as it is stored in **NODE_DETAILS** table in SLAC described above in **Archive site** section above. For this reason we designed a table for storing data of nodes that is previously placed in **nodes.cf** file and is used by different analysis and data collection scripts for their operation.

The structure of node_details table is shown below:

```
mysql> desc node_details;
+-----+-----+-----+-----+-----+
| Field          | Type                | Null | Key | Default |
+-----+-----+-----+-----+-----+
| id             | int(10) unsigned   | NO   | PRI | NULL     |
| nodename       | varchar(100)        | NO   |     | NULL     |
| ipaddress       | varchar(15)         | YES  |     | NULL     |
| sitename        | varchar(100)        | YES  |     | NULL     |
| nickname        | varchar(35)         | YES  |     | NULL     |
| country         | varchar(100)        | YES  |     | NULL     |
| continent       | varchar(100)        | YES  |     | NULL     |
| lat            | varchar(10)         | YES  |     | NULL     |
| lon            | varchar(10)         | YES  |     | NULL     |
| projecttype     | varchar(10)         | YES  |     | NULL     |
| pingserver      | varchar(100)        | YES  |     | NULL     |
| traceserver     | varchar(100)        | YES  |     | NULL     |
| dataserver      | varchar(100)        | YES  |     | NULL     |
| url            | varchar(100)        | YES  |     | NULL     |
| gmt            | varchar(10)         | YES  |     | NULL     |
+-----+-----+-----+-----+-----+
15 rows in set (0.01 sec)
```

Figure 12: Metadata of Node_Details Table

One of the important fields is lat/Lon which is used in calculation of ALPHA. Similarly projecttype tells either it is a monitoring or a remote site. And dataserver tells the URL for collecting raw data from this monitoring site.

The raw data for analysis which is gathered from monitoring sites by archive site contains host names, RTTs and number of packets sent, received etc. This data from different monitoring site is copied to a text file at archive site on daily basis. The file name is like **ping-yyyy-mm-dd.txt.gz** for example **ping-2011-03-04.txt.gzas** described above.

Inside this file the data is placed in the format as shown below

```
pinger.uet.edu.pk 111.68.102.14 waib.gouv.bj 81.91.232.2 100 1299198408 10 10
385.896 386.703 387.492 1 2 3 4 5 6 7 8 9 10
316 299 320 293 283 300 331 289 303 318
```

Figure 13: Data Format in file

Above are shown three parts of one line of data stored in this file. We will explain each field of this line one by one to understand data and how it can be placed into table.

The first part of this line as eight fields:

- Monitoring host name
- Monitoring host IP
- Remote host name
- Remote host IP
- Packet size
- Unix Epoch Time
- Packets sent
- Packets received

First three fields of second part of this line are:

- Minimum RTT

- Average RTT
- Maximum RTT

Remaining fields are sequence numbers of packets from 1-10. Similarly third part of line contains RTTs of respective sequence numbers of packets.

As this file is generated by archive site daily so it contains data for all monitoring sites for only one day. This file is read by the analysis script named **analyze-hourly.pl**, which is responsible for analysis of one day data i.e. it calculates different analysis metrics for every hour of that day and put the results in a file like **metric-packet size-by site or node-yyyy-mm-dd.txt.gz** for example **average_rtt-100-by-node-2011-03-01.txt.gz**.

Keeping in mind the format of data of these files we can to design a table that stores the data of these files in it so that only one table contain raw data for all times and there is no need of separate file every day. The table that is designed for that purpose is shown below in a figure:

```
mysql> use archive1;
Database changed
mysql> desc pingdata;
```

Field	Type	Null	Key	Default
monnode_id	int(11)	YES		NULL
remnode_id	int(11)	YES		NULL
packet_size	int(11)	YES		NULL
time	varchar(20)	YES		NULL
sent	int(11)	YES		NULL
recv	int(11)	YES		NULL
min_rtt	float	YES		NULL
avg_rtt	float	YES		NULL
max_rtt	float	YES		NULL
seq1	int(11)	YES		NULL
seq2	int(11)	YES		NULL
seq3	int(11)	YES		NULL
seq4	int(11)	YES		NULL
seq5	int(11)	YES		NULL
seq6	int(11)	YES		NULL
seq7	int(11)	YES		NULL
seq8	int(11)	YES		NULL
seq9	int(11)	YES		NULL
seq10	int(11)	YES		NULL
rtt1	float	YES		NULL
rtt2	float	YES		NULL
rtt3	float	YES		NULL
rtt4	float	YES		NULL
rtt5	float	YES		NULL
rtt6	float	YES		NULL
rtt7	float	YES		NULL
rtt8	float	YES		NULL
rtt9	float	YES		NULL
rtt10	float	YES		NULL
date	date	YES		NULL

```
30 rows in set (0.10 sec)
```

Figure 14: Metadata of pingdata

All the data which is placed in raw data file is stored in this table either directly or indirectly e.g. the date which is embedded with file name is stored in a field and only IDs for monitoring and remote hosts are stored in this table all the required information for every host is placed in **node_details** table.

Now we come to the analysis scripts i.e. **analyze-hourly.pl (15.3)**, **analyze-daily.pl (15.4)**, **analyze-monthly.pl (15.5)**, **analyze-yearly.pl**, **analyze-allyears.pl** etc. These scripts are changed in such a way that they read values from ping_data table and analyze accordingly. The results are then stored in the analysis table. Form analysis table, we can get the values of any metric according to all tick types i.e. hourly, daily and monthly etc. the structure of analysis table is described as

Analysis	
PK,FK1	<u>Monitoring host id</u>
PK,FK2	<u>Remote host id</u>
PK	<u>Year</u>
PK	<u>Month</u>
PK	<u>Day</u>
PK	<u>Hour</u>

By-Class
Packet size
Conditional Loss Probability
Minimum Round Trip Time
Ping Unreachability
Minimum Packet Loss
Ping Unpredictability
Inter Packet Delay Variation
Zero Packet Loss Frequency
Packet Loss
Duplicate Packets
Inter Quartile Range
Mean Opinion Score
Alpha

Figure 15: Metadata of Analysis Table

Here the problem arises that how to differentiate the data for different tick types i.e. how hourly data is differentiated from daily data as all aggregation and calculation cannot be done on runtime or at the time of request. So we should store daily and monthly aggregated data in the same table but in different pattern than hourly data which is basic unit of the Analysis table. For this the hour field is set to null while storing the daily data. We can easily get analyzed data for one month, last 60 or 120 days. Similarly to store monthly analysis data we set **Hour** as well as **Day** field to null so that for collecting data for certain number of months we select those

rows whose **Hour** and **Day** field is set to null. Similar is the case for yearly data in which we set the **Month, Day, Hour** fields of analysis table to null.

There are many advantages of this structure of analysis table as compared to flat-files. An example is given below:

The **analyze-daily.pl** script analyzes data on daily basis by collecting hourly analysis data from analysis table. This script executes every day and replaces the old file with current and updated file like **metric-packet_size-by_class-days-120.txt.gz** or **metric-packet_size-by_class-yyyy-mm.txt.gz**. As this script is executed daily so it analyzes data of today as well as of previous days, deleting the current file and making a new file every day. There is a lot of repetition in this process and also lots of calculations are done on data of previous days which has already been analyzed.

The **analyze-hourly.pl** script is also executed daily to analyze hourly data. But each time it is executed, it makes a new file for each day. **Analyze-daily.pl** on the other hand, deletes and reconstructs the same file for one month or 60 or 120 days. Deleting of this file means erasing all the analyzed data for previous days and again analyzing the data for previous days as well. This procedure results in lot of extra calculations every day.

But in the database architecture, the data is in proper manner and structured, therefore whenever a script is executed only required hourly analysis data is retrieved and only daily analysis for that day is performed, not for all days as in flat-files. This saves a lot of extra calculations and results in increase of efficiency. Similarly in case of monthly analysis, the data of only a specific month is analyzed instead of analyzing the data of each month every day. In the same way there is also no need to analyze the data for last 60 or 120 days as this can be easily retrieved from the same analysis table.

3.6 ADDING MOS AND ALPHA:

The motivation for adding these two additional metrics in archive site is that these two metrics are very useful in our daily life on internet connectivity and usage. Their detailed description is explained below but the major advantage of using these

metrics is that we monitor network performance in a totally different context and brings new perspective in understanding our data. VOIP is a vastly used technology and in the same way Geo Location gains much importance in recent years so adding these metrics in archive site contribute a lot to SEECS and SLAC in research of these fields.

Initially these two metrics were added in flat-file based architecture, then tested and deployed on SEECS PingER server. The whole previous raw data up till September 2009 was analyzed to calculate these metrics. After successful deployment these metrics are also added in our relational database architecture and we make two additional fields in analysis table as shown above.

Following are the detailed description of both of these metrics:

3.6.1 Mean Opinion Score (MOS):

The telecommunications industry uses the Mean Opinion Score (MOS) as a voice quality metric. The values of the MOS are: 1= bad; 2=poor; 3=fair; 4=good; 5=excellent. A typical range for Voice over IP is 3.5 to 4.2. In reality, even a perfect connection is impacted by the compression algorithms of the codec, so the highest score most codec's can achieve is in the 4.2 to 4.4 range. For G.711 the best is 4.4 (or an R factor (see ITU-T Recommendation G.107, "The E-model, a computational model for use in transmission planning.") of 94.3) and for G.729 which performs significant compression it is 4.1 (or an R factor of 84.3).

There are three factors that significantly impacts call quality: latency, packet loss, and jitter. Other factors include the codec type, the phone (analog vs. digital), the PBX etc.). Most tool-based solutions calculate what is called an "R" value and then apply a formula to convert that to an MOS score. We do the same. The R value score is from 0 to 100, where a higher number is better. Typical R to MOS values is shown below:

1. $R=90-100 \Rightarrow MOS=4.3-5.0$ (very satisfactory)

2. R=80-90=>MOS=4.0-4.3 (satisfactory)
3. R=70-80=>MOS=3.6-4.0 (some dissatisfaction)
4. R=60-70=>MOS=3.1-3.6 (more dissatisfaction)
5. R=50-60=>MOS=2.6-3.1 (Most dissatisfaction)
6. R=0-50=>MOS=1.0-2.6 (not recommended).

To convert latency, loss, jitter to MOS we follow Nessoft's method.

Pseudo code is shown below:

```

#Take the average round trip latency (in milliseconds), add
#round trip jitter, but double the impact to latency
#then add 10 for protocol latencies (in milliseconds).
EffectiveLatency = ( AverageLatency + Jitter * 2 + 10 )
#Implement a basic curve - deduct 4 for the R value at 160ms of
latency
#(round trip). Anything over that gets a much more aggressive
deduction.

ifEffectiveLatency< 160 then
    R = 93.2 - (EffectiveLatency / 40)
else
    R = 93.2 - (EffectiveLatency - 120) / 10

#Now let's deduct 2.5 R values per percentage of packet loss (i.e.

```

analyze-hourly.pl to calculate hourly values of these metrics. So from raw data this script also calculates Average RTT, IPDV (Jitter) and Packet Loss. When these required metrics are calculated by the script then at run time they are used in calculation of MOS.

3.6.2 ALPHA:

In tri-lateration and multilateration based Geolocation algorithms, the correct estimation of the location of any target node depends on the correct mapping of delay to distance. When we get RTT values from landmarks to the target node, the

next important step is to map this RTT value to a distance that can correctly represent the radius of the circle drawn around that particular landmark. Since, the overlapping area of the circles is used to estimate the location (latitude, longitude) of the target node; the correct estimation purely depends on the radius of circles drawn.

Thus we need to find a suitable correlation between the delay measurements and the distance values. There are various factors affecting the RTT values including: propagation delays; router forwarding and queuing delays; unavailability of great circle path; presence of satellite connections etc. These make it impossible or at least difficult to reach to a single common factor which could be used in the delay to distance mapping.

We know that digital information travels in fiber at a speed of 0.6 times the speed of light in vacuum. Thus we can say that 1ms of RTT can equal roughly 100Km distance. But in order to tackle the additive distortions in RTT values due to the various delaying factors mentioned above, use of this 100Km/ms alpha value results in a large over estimation. As a result many geolocation location techniques, such as Octant, Constraint Based Geolocation, and Topology Based Geolocation use much smaller values of 40-60 Km/ms for alpha.

Our goal is to find values of alpha that can more accurately map RTT values to geographical distance. So the formula for calculation of ALPHA is give below:

'c' = speed of light in vacuum i.e. 299,792,458 m/s

Using 300,000 km/s as 'c' this yields:

$$\mathbf{RTD [km] = Alpha * min_RTT [ms] * 100[km/ms]}$$

Alpha is a way to derive Round Trip Distance (RTD) between two hosts (using minimum RTT).

Or if we know the RTD then we can derive Alpha

$$\mathbf{Alpha = RTD [km] / (min_RTT[ms] * 100[km/ms])}$$

Alpha is a way to measure directness of internet routes. Large values of Alpha close to one indicate a direct path. Small values usually indicate a very indirectly routed path. This assumes no queuing and minimal network device delays.

Round Trip Distance (RTD) which is used in calculation of ALPHA is calculated at run time by obtaining the Lat, Lon of two hosts. From these Lat, Lon we calculate distance between these two nodes and double this distance to calculate RTD for these two nodes for which ALPHA needs to be calculated. In flat-files system these Lat, Lon are obtained from **nodes.cf** file and in database architecture Lat, Lon are obtained from **node_details** table which is basically a table replica of **nodes.cf** file.

4 RESULTS

The results of analysis are shown on browser by a script named **pingtable.pl**. Previously in flat-files this script used to read data from analysis files which were generated by various analysis scripts at archive site. The data that is read from analysis files is formatted by a script named **mon-lib.pl** and display the results on browser. Results for every metric at any interval of time and for any pair of nodes are available in it. But in database architecture as all the analysis results are stored in single table so we changed the script **pingtable.pl** such that it reads values from table by executing SQL queries for results of different metrics at any interval of time and for any pair of nodes. Below are the results that are shown in browser after modifying **pingtable.pl**.



PingER Site-by-daily History Table for TCP Throughput (kbits/s) by-node with 100 byte packets to Pakistan seen from Pakistan excluding none



Make selections on current dataset

Metric By For Packet Size (bytes) Tick-Type For Year Month

From To Excluding

Show only Beacon sites Change the dataset Only show sites with datapoints. Filter Data (monthly, allmonthly & allyearly data only. Uses /home/pinger/metadata/pingtable_filter.cfg)

Throughput	Click on	Please Note
more than 1000kpbs are shown black . 1000kpbs to 500kpbs are shown green . 500kpbs to 100kpbs are shown orange . 100kpbs to 50kpbs are shown pink . 50kpbs or less are shown red .	Each Column Header to sort by that column Monitoring Site to graph the data with (only for SLAC monitoring host & By=by-node) Smokeping Remote Site to jump to the graphing facility at FNAL ? to get location information for that Remote_Site from the database	The values shown here are for the full day GMT time.

<u>Monitoring-Site</u>	<u>Remote-Site</u>	<u>?</u>	<u>11May01</u>	<u>11May02</u>	<u>11May03</u>	<u>11May04</u>	<u>11May05</u>
PK.KOHAT.EDU.N2	PK.QUETTA.PERN.EDU.N2	?M	670.192	555.042	492.407	330.751	354.594
PK.KOHAT.EDU.N2	PK.COMSATS.EDU.N2	?M	2166.829	2025.851	669.384	283.009	800.153
PK.KOHAT.EDU.N2	PK.GIKI.EDU.N1	?M
PK.KOHAT.EDU.N2	PK.NCP.EDU.N3	?M	417.942	455.844	324.880	323.143	425.949
PK.KOHAT.EDU.N2	PK.PERN.RCP.EDU.N2	?M	14856.167
PK.KOHAT.EDU.N2	PK.AIRUNIVERSITY.EDU.N2	?M	416.895	429.452	334.377	313.574	366.397
PK.KOHAT.EDU.N2	PK.CEMB.EDU.N2	?M	239.912	1341.094	727.298	456.349	517.759
PK.KOHAT.EDU.N2	PK.HU.EDU.N2	?M
PK.KOHAT.EDU.N2	PK.PERN.EDU.N1	?M	837.321	536.963	403.217	277.630	214.195

Figure 16: Result Output

Above figure shows a fragment of result of TCP Throughput. Similarly all the metrics can also be displayed including MOS and ALPHA. The output remains same for both flat-files and database architecture as shown above. Only the mechanism of collection of values is different for these approaches.

Now we are going to display the data and results that are stored in tables instead of flat-files that are shown earlier so that we can take a look of database to

understand working of database architecture archive site. Some of values in **node_details** table are shown below in the figure:

```
gmt: N/A
***** 150. row *****
id: 1986
nodename: pinger-rcp.pern.edu.pk
ipaddress: 121.52.150.231
sitename: rcp.pern.edu.pk
nickname: PK.PERN.RCP.EDU.N2
country: Pakistan
continent: South Asia
lat: 33.9912
lon: 71.4428
projecttype: M
pingserver: http://pinger-rcp.pern.edu.pk/cgi-bin/traceroute.pl?function=ping
traceserver: http://pinger-rcp.pern.edu.pk/cgi-bin/traceroute.pl?
dataserver: http://pinger-rcp.pern.edu.pk/cgi-bin/ping_data.pl?
url: http://pinger-rcp.pern.edu.pk/
gmt: N/A
***** 151. row *****
id: 1991
nodename: pinger.cdacmumbai.in
ipaddress: 202.141.151.30
sitename: cdacmumbai.in
nickname: IN.CDACMUMBAI.N1
country: India
continent: South Asia
lat: 18.93
lon: 72.85
projecttype: MB
pingserver: http://pinger.cdacmumbai.in/cgi-bin/traceroute.pl?function=ping
traceserver: http://pinger.cdacmumbai.in/cgi-bin/traceroute.pl?
dataserver: http://pinger.cdacmumbai.in/cgi-bin/ping_data.pl?
url: http://www.cdacmumbai.in
gmt: N/A
***** 152. row *****
id: 1994
```

Figure 17: Node_Details table

Similarly of values stored in **pingdata** table which stores the raw data used for analysis is shown below:

```

        rtt8: 415
        rtt9: 416
        rtt10: 415
        date: 2011-04-27
***** 2. row *****
monnode_id: 10
remnode_id: 1691
packet_size: 100
        time: 1303946303
        sent: 10
        recv: 10
min_rtt: 230.995
avg_rtt: 231.223
max_rtt: 231.559
        seq1: 1
        seq2: 2
        seq3: 3
        seq4: 4
        seq5: 5
        seq6: 6
        seq7: 7
        seq8: 8
        seq9: 9
        seq10: 10
        rtt1: 231
        rtt2: 231
        rtt3: 231
        rtt4: 231
        rtt5: 231
        rtt6: 231
        rtt7: 230
        rtt8: 231
        rtt9: 231
        rtt10: 231
        date: 2011-04-27
***** 3. row *****
monnode_id: 10
remnode_id: 1646
packet_size: 100
        time: 1303946302
        sent: 10
        recv: 10
min_rtt: 309.517

```

Figure 18: Pingdata Table

In the same way the fragment of values stored in **analysis** table which stores the analysis results is shown below:

```

                iqr: 1.121
                average_rtt: 15.185
                throughput: 769.180
    out_of_order_packets: 14.286
                mos: 4.880
                year: 2011
                month: 4
                day: 3
                hour: NULL
***** 309. row *****
                mon: pingermtn.pern.edu.pk
                rem: monitor.niit.edu.pk
                pksize: 100
                byclass: by-node
conditional_loss_probability: 14.286
                minimum_rtt: 13.816
                ping_unreachability: 14.286
                minimum_packet_loss: 0.000
                ping_unpredictability: .
                ipdv: 1.305
                zplf: 14.286
                packet_loss: 100.000
                alpha: 1.350
                duplicate_packets: 0.840
                iqr: 1.121
                average_rtt: 15.185
                throughput: 769.180
    out_of_order_packets: 14.286
                mos: 4.880
                year: 2011
                month: 4
                day: 2
                hour: NULL
***** 310. row *****
                mon: pingermtn.pern.edu.pk
                rem: monitor.niit.edu.pk
                pksize: 100
                byclass: by-node
conditional_loss_probability: 14.286
                minimum_rtt: 13.816
                ping_unreachability: 14.286
                minimum_packet_loss: 0.000
                ping_unpredictability: .
                ipdv: 1.305

```

Figure 19: Analysis Table

Now we come towards the MOS and Alpha. These two matrices are added in flat files based architecture of PingER and working on <http://pinger.seecs.edu.pk/cgi-bin/pingtable.pl>.

5 DISCUSSION AND RECOMMENDATION

Now we come towards the discussion on the results and some recommendations to overcome the current challenges.

5.1 DISSCUSSION

The archive site based on database architecture has advantages like data is manageable, scalable and organized. Moreover, there is no need to calculate results from thousands of files, instead only one query is now required. But besides these advantages, there are some drawbacks also, e.g. more time is taken by scripts for collecting and analyzing data as compared to flat-files and database tables acquire more space than flat-files. However, this holds little significance as compared to the advantages offered since most of the analysis is done on daily, monthly or yearly basis. Therefore, data analysis can afford to be slow as it does not take place at runtime/real-time.

5.2 RECOMMENDATIONS:

However, there is always room for improvement and this is an interesting case academically. One relatively quick way of improving speed of the analysis is to change how scripts perform calculations. For example, doing most of the calculations and keeping processed data while also storing raw data. In theory, this should improve performance since retrieval will not require table joins or any involved iterations. Performance can also be achieved by caching commonly used queries. One solution is that we should change the schema and keep it closer to already deployed database schema of another network performance monitoring infrastructure, e.g. **PerfSonar** etc. which is the future work of this project, i.e. integrating **PingER** schema with **PerfSonar** to get a common interface of both infrastructures.

6 CONCLUSION:

All the research done on archive site flat-files system and conversion of this system to database architecture can be useful as it provides insight to many hidden issues of archive site analysis. This new database based system makes the archive site extensible in future. This is also a step towards an update of archive site which is a created a decade ago and needs to be updated according to the requirements of modern world (Technology change). This research also opens doors for advanced research in this field and also helps in adding new technologies in current infrastructure and also help in integrating this infrastructure with another to provide flexibility and improvements in network performance monitoring by covering more areas of the world by covering all areas of world.

II. Part II

Integration of TULIP with CBG

7 INTRODUCTION AND BACKGROUND

7.1 GEOLOCATION^[9]

Geolocation is the identification of the real-world geographic location of an object, such as radar, mobile phone or an Internet-connected computer terminal. Geo location may refer to the practice of assessing the location, or to the actual assessed location. The world is already into a transition from relatively fixed computers to mobile devices. This can potentially open a gateway for accurate locating services that can have multiple usages.

7.2 IMPORTANCE OF GEOLOCATION

The Internet has become a collection of resources meant to appeal to a large general audience. Although this multitude of information has been a great boom it also has diluted the importance of geographically localized information. Offering the ability for Internet users to gather information based on geographic location can decrease search times and increase visibility of local establishments. Similarly, user communities and chat-rooms can be enhanced through knowing the locations (and therefore, local times, weather conditions and news events) of their members as they roam the globe. It is possible to provide user services in applications and Web sites without the need for users to carry Global positioning system (GPS) receivers or even to know where they themselves are. Geo location by IP address is the technique of determining a user's geographic latitude, longitude and, by inference, city, region and nation by comparing the user's public Internet IP address with known locations of other electronically neighboring servers and routers.

IP addresses are not static - they can be reassigned, relocated within the same provider or being forwarded using mechanism like virtual private network systems

(VPNS) or other tunnels. A database consisting of geo locational information about IP hence has to be updated and maintained on a regular base. Due to the large and increasing number of used IP addresses, this is a task which is nearby impossible to be maintained manually. A stripped down version of their commercially available database is given out for free with several tools and API's, and is quite popular within the open source community.

In recent years many efforts have been made for IP geo-location. Very few are based on dynamic measurements.

7.3 TYPES OF GEOLOCATION

There are different types of geolocation algorithms that differ in their mechanism

7.3.1 CBG

CBG (Constraint Based Geo location) transforms delay measurements to geographic distance constraints, and then uses multilateration to infer the Geo location of the target host. CBG is able to assign a confidence region to each given location estimate. This allows a location-aware application to assess whether the location estimate is sufficiently accurate for its needs.

7.3.2 GeoIP

This technique uses end users input. It acquires data from various sites that require users to register themselves for service. Based on this data it estimates the location of an ASN. This technique is ideal for e-commerce applications but fails for intermediate and backbone routers. This is observed mainly because routers belonging to one AS may be in different countries. Hence for scientific study GeoIP will not be the ideal tool. E.g. if we are to find the geographical location of a bottleneck it may be an intermediate router.

7.3.3 Domain Name Services

DNS may also help in locating a host. The DNS LOC (location) resource record is designed to make this data available. In addition the names of routers often contain their location (e.g. city) so a trace route may help identify where a host is near.

7.3.4 Autonomous system

Given an IP or host name you can use Fixed Orbits to find the relevant AS. Then using a table of AS number to name you can find out more about the AS (e.g. contacts, HQ site etc.)

7.4 TULIP^[7]

TULIP stands for Trilateration Utility for Locating IP hosts. TULIP was developed by NUST-SEECs in collaboration with SLAC under the Internet End-to-End Performance Monitoring (IEPM) project. TULIP's purpose is to geo locate a specified target host (identified by host name or IP address) using RTT delay measurements of ping to the target from reference landmark hosts whose positions are well known. Estimating speed of light in copper and fiber, latitude and longitude of the target is calculated with respect to given landmarks. TULIP calculates the latitude and longitude coordinates (LAT/LON) of landmarks with the help of reflector.cgi and sites.xml.

7.4.1 Reflector.cgi

This is a CGI script placed at SLAC server. When it runs it makes all active landmarks to ping all active targets and collects RTTs into a file.

7.4.2 Land marks

Accuracy of the TULIP algorithm increases with increase in number of landmarks and the fact how well they are spread out over a continent. The network infrastructure

to North America and East Asia is similar and well structured. Other parts of the world such as Africa lack a structured infrastructure. In such cases quantifying a correlation between distance and RTT is next to impossible .with present techniques more landmarks in developing and under-developed countries with help in improving accuracy of locating targets. We have three main types of landmarks:

- PingER landmarks
- Planet lab landmarks
- PerfSONAR landmarks

7.4.3 Sites.xml

This file is placed at SLAC server. From this file we collect LAT/LON of the Landmarks and also other information like city, country and continent name.

The format of sites.Xml file looks like

```

- <nmtb:location>
  <nmtb:institution>Stanford Linear Accelerator Center</nmtb:institution>
  <nmtb:continent>North America</nmtb:continent>
  <nmtb:country>United States</nmtb:country>
  <nmtb:city>Menlo Park</nmtb:city>
  <nmtb:longitude>-122.203</nmtb:longitude>
  <nmtb:latitude>37.4177</nmtb:latitude>
</nmtb:location>
- <nmtl3:port id="urn:ogfnetwork:domain=slac.stanford.edunode=port=134.79.197.197">
  <nmtl3:ipAddress type="IPv4">134.79.197.197</nmtl3:ipAddress>
</nmtl3:port>
</nmtb:node>
</nmtb:domain>
- <nmtb:domain id="urn:ogfnetwork:domain=tenet.ac.za">
  - <nmtb:node id="urn:ogfnetwork:domain=tenet.ac.zanode=brunsvigia">
    <nmtb:name type="string">brunsvigia</nmtb:name>
    <nmtb:hostName>brunsvigia.tenet.ac.za</nmtb:hostName>
    <nmtb:description/>
  - <pinger:serviceInterface type="PingER">
  - <pinger:pingURL>

```

Figure 20: Sites.xml

Note this is the information of only one target.

7.5 MOTIVATION FOR INTEGRATION

SLAC maintains a database for more than 540 landmarks from three different international infrastructures (PlanetLab, PingER and PerfSONAR), which cover over 99% of the world's internet population. The data that is collected by these infrastructures is used by all geolocation techniques. CBG uses this data to estimate the geographical location of some target, therefore integration of CBG has become the matter of high priority because it will enable us to display more accurate results for targets and consequently use this infrastructure to the best of its ability. Furthermore this will provide an amalgamation of best known Geolocation techniques for research and education purposes. This can also be a very useful tool for pursuing future projects in locating IP hosts. This can potentially open a gateway for accurate locating services that can have multiple uses.

CBG algorithm is currently implemented in Matlab code. Whereas TULIP and Apollonius deployment at SLAC is Java based. We will integrate the two such that TULIP and Apollonius will keep running at SLAC and there'll be a Matlab server at SEECS running CBG code. There are two things which we need to take care of:

1. Integration should be transparent i.e. the end user will execute the TULIP GUI as before without any changes.
2. Integration should introduce minimum possible changes into the current TULIP architecture.

7.6 PROBLEM STATEMENT

There are many existing algorithms like Apollonius, TBG, and Trilateration etc. being used for Geolocation of IP hosts. But results of these algorithms are not much accurate e.g. in case of Apollonius the accuracy is only 30% and in case of TULIP the accuracy is 50%. Our goal is to achieve maximum accuracy. CBG is found to be relatively much more accurate than other existing algorithms including Trilateration,

Apollonius and TBG. In case of CBG the accuracy is 70% and due to its greater accuracy we want to integrate it with TULIP

8 LITERATURE REVIEW

8.1 CONSTRAINT BASED GEOLOCATION

Previous work on the measurement-based Geolocation of Internet hosts uses the positions of reference hosts with well-known geographic location as the possible location estimates for the target host. This leads to a discrete space of answers, i.e. the number of answers is equal to the number of reference hosts, which can limit the accuracy of the resulting location estimation. This is because the closest reference host may still be far from the target.

To overcome this limitation, we propose the Constraint-Based Geolocation (CBG) approach, which infers the geographic location of Internet hosts using multilateration. Multilateration refers to the process of estimating a position using a sufficient number of distances to some fixed points. As a result, multilateration establishes a continuous space of answers instead of a discrete one.

Constraint Based Geolocation (CBG) is an algorithm that is used for calculating geographical location (Geolocation) of IP hosts (each host is called a target). This technique takes multiple landmarks (hosts whose lat/lon coordinates are known) as input to plot circles and identify possible regions of intersection of these circles. Each landmark is a center of a circle and the radius of a circle is the distance from a landmark to the target. The radial distance of each circle is calculated from the minimum Round Trip Time (RTT) between a landmark and the target. RTTs are gathered from pings.

CBG works on three Constraints which are given below

- Speed of light
- Speed of internet
- Bestline

This technique takes multiple landmarks (hosts whose LAT/LON coordinates are known) and ping the targets to plot circles and identify possible regions of intersection of these circles. CBG draws two circles instead. One is called baseline and other is called bestline. Baseline is calculated assuming no distortion in the links while bestline considers all parameters like processing time and queues on routers etc. along with calculated RTTs in ideal conditions. Actually bestline is result of the actual geographic distance plus an additive geographic distance distortion while baseline is only real geographic distance calculated. These two lines form shapes of donuts intersecting each other. The area of intersection in this case is much reduced as compared to intersection area of single circles. This leads in more precise estimation of target. Fig 1 shows donut shapes formed by three landmarks L1, L2, L3. Baseline is denoted by solid line where as bestline is denoted by dotted line. Each landmark is a center of a circle and the radius of a circle is the distance from a landmark to the target. The radial distance of each circle is calculated from the minimum Round Trip Time (RTT) between a landmark and the target. RTTs are gathered from pings.

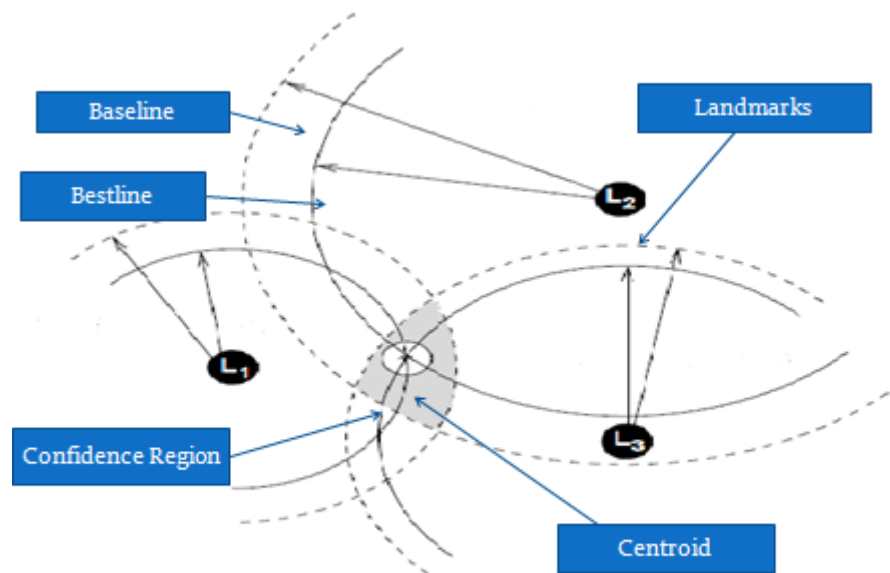


Figure 21: CBG Mechanism

The first version of CBG code was independently runnable. It was necessary to modify CBG code to take input from already running TULIP. We had to modify CBG code in such a way that it could be executed for a single target instead of all targets and that target should be user defined. Landmarks that will ping the target can also be specified at runtime. This modified code calculates results for single target and stores in a file.

After modification of CBG code, TULIP code was also modified in such a way that LAT/LON coordinates of landmarks and RTT's provided by reflector.cgi were written to a file. The data written in this file was used as input to CBG code.

CBG code has three techniques with respect to constraints. Here is input format of each technique.

Input to speed of light (SOL) constraint

For **cbg2.m** (which has constraint of speed of light) input is like

```

Target_lat    Targer_lon    Target_id      0
Landmark1_lat    Landmark1_lon    Landmark1_id    RTT
Landmark2_lat    Landmark2_lon    Landmark2_id    RTT
Landmark3_lat    Landmark3_lon    Landmark3_id    RTT

```

Here the target is pinged by the given landmarks and RTT mentioned is the minimum RTT of each landmark with the target.

Input to speed of internet (SOI) constraint

First Target entries:

```

Target_lat    Targer_lon    Target_id      0

```

Landmark1_lat	Landmark1_lon	Landmark1_id	RTT
Landmark2_lat	Landmark2_lon	Landmark2_id	RTT
Landmark3_lat	Landmark3_lon	Landmark3_id	RTT

Second Input number of landmarks

Third input provides entries of each landmark:

Landmark _lat	Landmark_lon	Landmark _id	0
Target1_lat	Target 1_lon	Target 1_id	RTT
Target 2_lat	Target 2_lon	Target 2_id	RTT
Target 3_lat	Target 3_lon	Target 3_id	RTT

Where above is the landmark and targets pinged by this landmark and the RTT is the minimum RTT of each target with the landmark.

The output to the CBG code like this:

Target_id	est_loc_lat	est_loc_lon	act_loc_lat	act_loc_lon
err_km	region_area_km^2	dist_nearest_landmark		
constraint_type	in_region (1 or 0)			

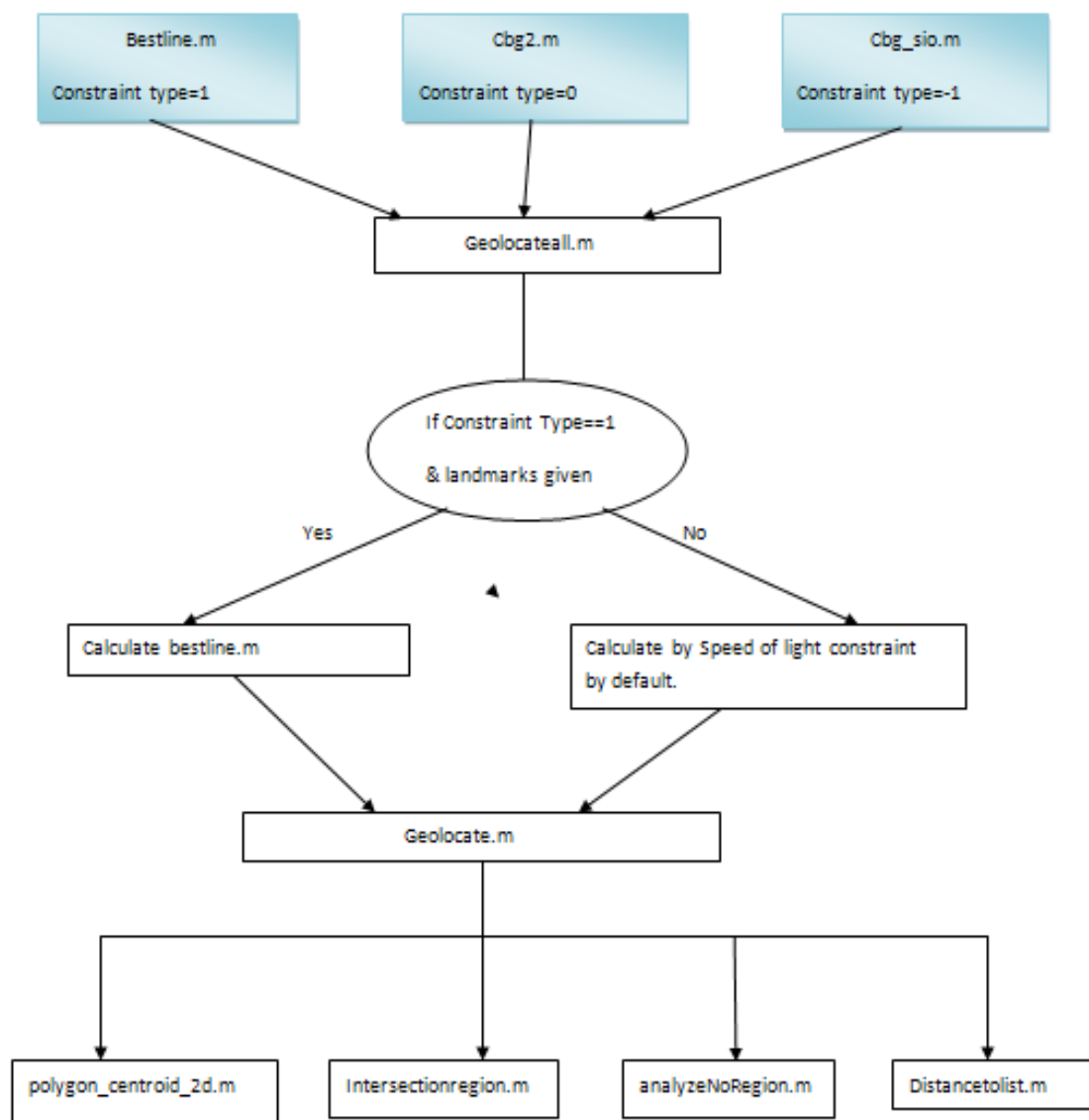


Figure 22: CBG Flow Diagram

9 METHODOLOGY

The TULIP and CBG integration is done in such a way that TULIP is running as java client, on the other hand, CBG code is running on the Matlab Server. The architecture of integration is given below.

9.1 ARCHITECTURE OF INTEGRATION

- 1) User enters the URL with target to be geolocated.
- 2) The TULIP visualization GUI on the client calls the Java code. The Java code calls the reflector RTT server with the target and information on the selection of landmarks. The reflector calls the enabled landmarks from the set of selected landmarks. The landmarks measure the ping RTTs to the target and return the results of the pings to the reflector that in turn returns those results to the client. The client then massages the results and computes the answer. Meanwhile the client sends a query string via a web service/remote routine (Perl or CGI script) to the Matlab server. Detail below:
 - i) The TULIP client gets the active landmarks list from the reflector as before.
 - ii) Client massages/adjusts the results and computes the answer for the target that needs to be geolocated.
 - iii) Client needs to send this active landmark and target list to Matlab as well. To accomplish this, the client (which is written in Java) writes a file.
 - iv) A perl script can then use sockets to establish a connection to the Matlab server and send this file.
 - v) Another perl script running on the Matlab server receives the file via sockets and writes it in local directory.

- vi) Matlab CBG code then needs to read this file and provide the contents as input to the geolocateall method.
- 3) The Matlab server computes an answer from data provided in the query string. Matlab server sends back a reply in the form of a dataset. Detail below:
 - a) Matlab CBG code computes an answer from the list of landmarks and target.
 - b) Matlab CBG code writes the result to a file.
 - c) This file is sent to the TULIP client using perl script. This will be the same script running on Matlab server which was used to receive the active landmarks and target list.
 - d) The file will be received at TULIP client by perl script.
 - e) The TULIP Java code will be modified to read this file and put the CBG results on the map.
 - 4) The Java code receives the reply and merges this to other results to create a standardized XML file.
 - 5) This XML file is then read by TULIP visualization GUI to display the result on Google map. Each algorithm's result is represented in a distinct fashion

Tulip-CBG Integration

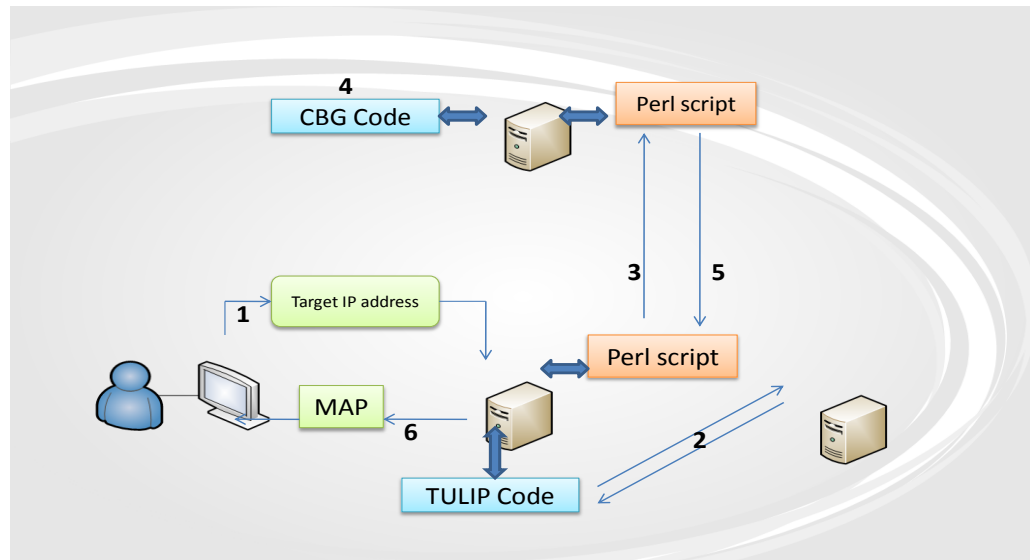


Figure 23: TULIP CBG Integration

9.2 TOOLS USED

The tools that are used in implementing the integration of TULIP-CBG are:

1. Java (jdk1.6)
2. Matlab
3. Perl Scripting Language
4. Google Maps API

9.3 TULIP CONFIGURATION AT SEECS

We deployed TULIP at SEECS and replicated the exact same setup present at SLAC as it is deployed at SLAC. The software's and services required for the deployment of TULIP are explained below:

9.3.1 Software and Services

1. Java (jdk 1.6)
2. Ant (1.7)
3. Apache
4. Google Map API

We deployed TULIP on PerfSONAR server at SEECS. It requires a public IP because it collects landmarks data from reflector.cgi that is placed at SLAC. It also requires root access due to its directory structure and permission. All core classes are placed at `$TULIP_HOME/src/tulip/core/` and all related classes are placed at `$TULIP_HOME/src/tulip/util/`.

3.1.1 Running Procedure

The following commands are executed to run the code properly

- 1 **ANT** for compiling java code.
- 2 then we moved to `$TULIP_HOME/build` directory and then give the following command i.e. :

```
Java -cp commons-httpclient-3.1.jar:commons-logging-1.1.1.jar:commons-codec-1.3.jar:. Tulip.core.AutomateTest target 132.227.74.51 startNewTest
```

Java command remains as it is. We only change the respective target.

9.4 MATLAB SERVER CONFIGURATION AT SEECS

1. Install Matlab on Linux terminal..
2. Run `cbg_soi.m` for constraint based on Speed of Internet (SOI), which is here defined as the speed of optical signals in fibre or $\sim 2/3$ the Speed of Light in vacuum (c).
3. Run `bestline.m` for constraint based on bestline approach. The bestline is defined in the Constraint-Based Geolocation paper, but basically is the tightest fit over all the (delay, distance) pairs meant to never underestimate the distance for a delay.

9.5 INTEGRATION METHODOLOGY WITH TULIP

For Integration of TULIP with CBG we have changed the CBG code such that it reads an input file and the results is printed to an output file. On the other hand we have also changed TULIP code such that all the landmarks data i.e. their Lat/Lon/RTT (which the TULIP code also uses for its Trilateration calculations) is written to a file by TULIP code. Actually this written file is sent to CBG server as an input file as described above. By processing this input file the output file which is generated by CBG is send back to TULIP server. At this time another change in TULIP code takes place which reads this output or CBG result file and shows these results on map.

Now we explain the exchange mechanism of these input and output files between TULIP and CBG server. We have written three Perl scripts for this purpose. Two scripts are placed at the TULIP server and one on the CBG server. Two scripts that are on TULIP server are:

1. **Server.pl (15.11)**
2. **Client.pl (15.13)**

After collecting landmarks data into a file, TULIP code connects to the CBG server and transfers this file over the network by using the TCP protocol

CBG server receives this input file and calls CBG code with the received file as input parameters. CBG code processes this input into a TULIP compatible output. This out output is also saved to file and returned by the CBG server to the TULIP clients as a TCP response. TULIP while generating results for the other algorithms receives this file and plots the results on map, shown to user via a browser.

Currently integration of TULIP-CBG is running on speed of internet constraint in which speed of internet is used for making circles or Donuts which overlap with each other to calculate the confidence region in which CBG has confidence that a particular target must be placed in that region. Other constraint is speed of light which is also operational and the third constraint bestline will be discussed later.

Now we will explain the format of input file and Result file. In input file according to CBG code first line contains actual Lat, Lon, id and 0 of the target. These actual Lat, Lon are used for calculating error difference from estimate Lat, Lon but here actual Lat, Lon are not available as they are to be calculated so we put 0 there. Id is always set to 1 because there is no need of separate Id in this integrated system. Separate ids are only needed when we run CBG code for multiple targets. But in this case, we are executing for single target. All other lines contain Lat, Lon, RTT and id of landmarks that ping the given target and which are used for both TULIP and CBG calculation.

The output or result file contains id, actual Lat, Lon (which are set to 0), estimate Lat, Lon (which are calculated), error difference (difference between actual and estimate location), region area (confidence area in which target is located), in region (is the actual target placed in that region), hull bull (0 if target is to be located between convex hull of landmarks and 1 if to be locate target anywhere).

9.6 MULTITHREADING

Basically the TULIP and CBG code is multithreaded and many users can run the code at same time. The whole integration process also supports multiple users in such a way that there is no intermingling of values takes place in input and output files as their names depend upon their target names. The integration is transparent as the CBG Matlab code is running on backend same output as TULIP.

On both TULIP and CBG server's there are two directories. One directory is inputs and second is Results. Inputs directory contains input files whose names are given as target names i.e. if target is 10.3.20.10 then in inputs directory name of input file for that target is 10.3.20.10.txt. Similar is the case for output files. In results directory the result of target 10.1.11.11 is placed in output directory 10.1.11.11.txt. These names are given to input and output files automatically by code depending upon on which target the code is running for.

9.7 TULIP VISUALIZATION^[11]

Now we come towards GUI of TULIP-CBG integrated system. Before this we test integration on command terminal. There are three main components which are used for displaying results on GUI

1. **Test_table.html**
2. **Tulip_map.html**
3. **Tulip_viz.cgi**

Here the main component is tulip-viz.cgi and other two html files are called in this script when it is running on the browser. Basically tulip-viz.cgi runs the main code of tulip and the output of this tulip code is served as function call for add row () function which is present in test_table.html and used to display a table of landmarks information on browser. The map and the markers on the map, which identifies the landmarks and calculated target locations on map, are displayed by tulip_map.html. We have changed this file to include a CBG marker to also show result of CBG with already deployed TULIP, Apollonius and GeoIP. Due to a java script problem it was not operable on MSIE .However the problem has been fixed

There are some screens shots of result Map are given below.



Figure 24: TULIP Interface before CBG Integration

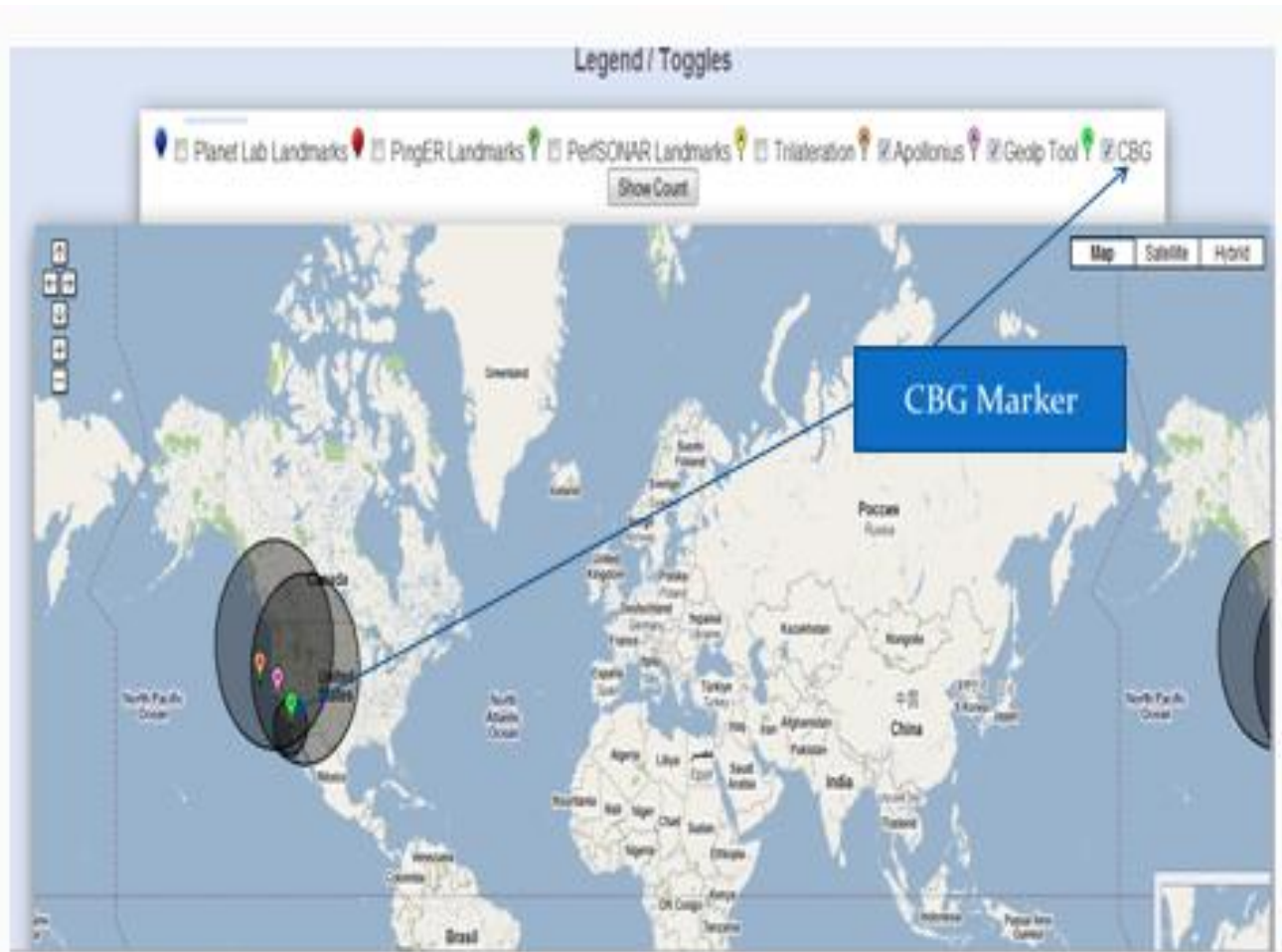


Figure 25: TULIP Interface after Integrating CBG

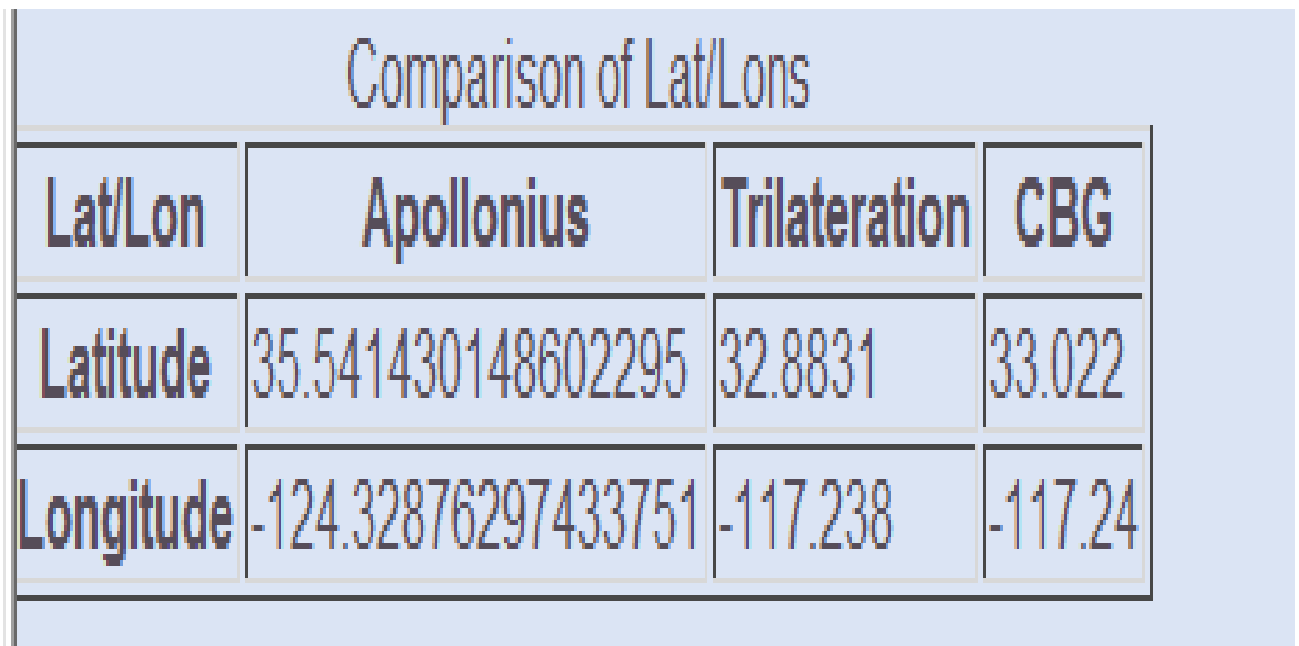
10 RESULTS AND PERFORMANCE EVALUATION

10.1 RESULTS

The results of CBG TULIP integration which are shown on browser are explained below

10.1.1 Comparison of Lat/Lon

When all the active landmarks calculate their Min RTT, Max RTT, Average RTT and distance to given target, Tulip, CBG and Apollonius calculate their Lat/Lon results and these results are displayed into a table. This helps in easy comparison of the estimated Lat/Lon calculated by Apollonius, Trilateration and CBG. The figure is given below



Lat/Lon	Apollonius	Trilateration	CBG
Latitude	35.541430148602295	32.8831	33.022
Longitude	-124.32876297433751	-117.238	-117.24

Figure 26: Results Table

The resulted Lat/Lon is also drawn on map to identify the position of target globally.

For better understanding below there is a figure of map without landmarks.

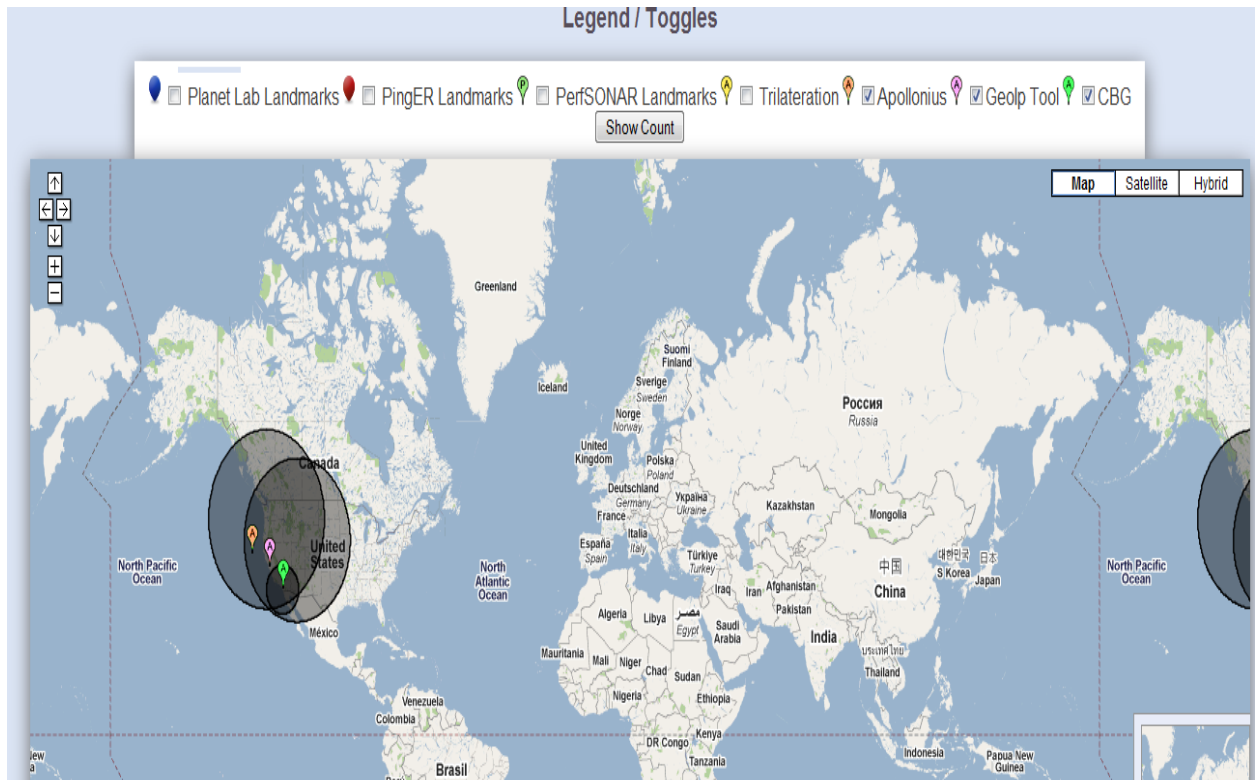


Figure 27: Without Landmarks

10.1.2 Ideal Case

If we give actual landmark as target IP address then estimated distance from that landmark would be 0 and both techniques CBG and Trilateration would give exact Lat/Lon, it means that we are testing techniques for best case (as distance from one point to target would be 0), so this means that accuracy is hundred percent but if we test it for any IP that is not a landmark then these techniques give different results because in this case distance is never zero from any point to target. Below there is an example in which we give the different landmarks as a target IP address and their accuracy come 100%..For example we select different city from North America

continent and we get the actual Latitude and Longitude from the Sites.Xml and then compare the results with CBG and Trilateration.

Table 5: Ideal Case

City	Country	Target	Lat/Lon	Actual lat/lon	CBG	Trilateration	Apollonius
Williamsburg	United States	128.239.22.9	Latitude	37.2707	37.271	37.2707	35.933
			Longitude	-76.7075	-76.707	-76.7075	-74.268
Houston	United States	192.124.228.18	Latitude	29.7629	29.763	29.763	31.8522
			Longitude	-95.383	-95.383	-95.383	-97.2877
Eagle Nest	United States	67.230.207.3	Latitude	36.5538	36.554	36.5538	41.323
			Longitude	-105.264	-105.26	-105.264	-109.107
Cambridge	United States	140.247.197.204	Latitude	42.3824	42.3824	42.3824	40.837
			Longitude	-71.0997	-71.099	-71.0097	-73.497

10.2 ACCURACY OF DIFFERENT CONTINENTS

10.2.1 Accuracy of Asia

Table 6: Accuracy in Asia

Country	Target	Lat/Lon	GeoIP	CBG	Trilateration	Apollonius
Korea	116.89.165.133	Latitude	24.788	25.051	22.3361	35.331
		Longitude	115.33	115.54	114.264	130.803
Taiwan	140.112.42.159	Latitude	25.0392	24.784	24.7812	21.507

		Longitude	121.525	120.99	120.993	136.026
India	120.88.46.30	Latitude	18.975	18.987	18.98	20.574
		Longitude	72.8258	72.83	72.83	81.5722
China	202.112.0.35	Latitude	39.9289	39.913	39.9092	38.835
		Longitude	116.3883	116.24	116.24	112.875
China	61.175.163.41	Latitude	30.2553	41.82	39.909	17.10333
		Longitude	120.1689	116.24	116.24	120.362
India	59.165.131.15	Latitude	18.975	18.98	18.98	24.7679
		Longitude	72.8258	72.83	72.83	83.77
Sri Lanka	192.248.48.3	Latitude	7.2631	19.38	18.93	8.772
		Longitude	80.6028	72.85	72.85	65.457
Sri Lanka	192.248.1.164	Latitude	7.2964	14.129	18.93	9.2207
		Longitude	80.635	77.58	72.85	66.54

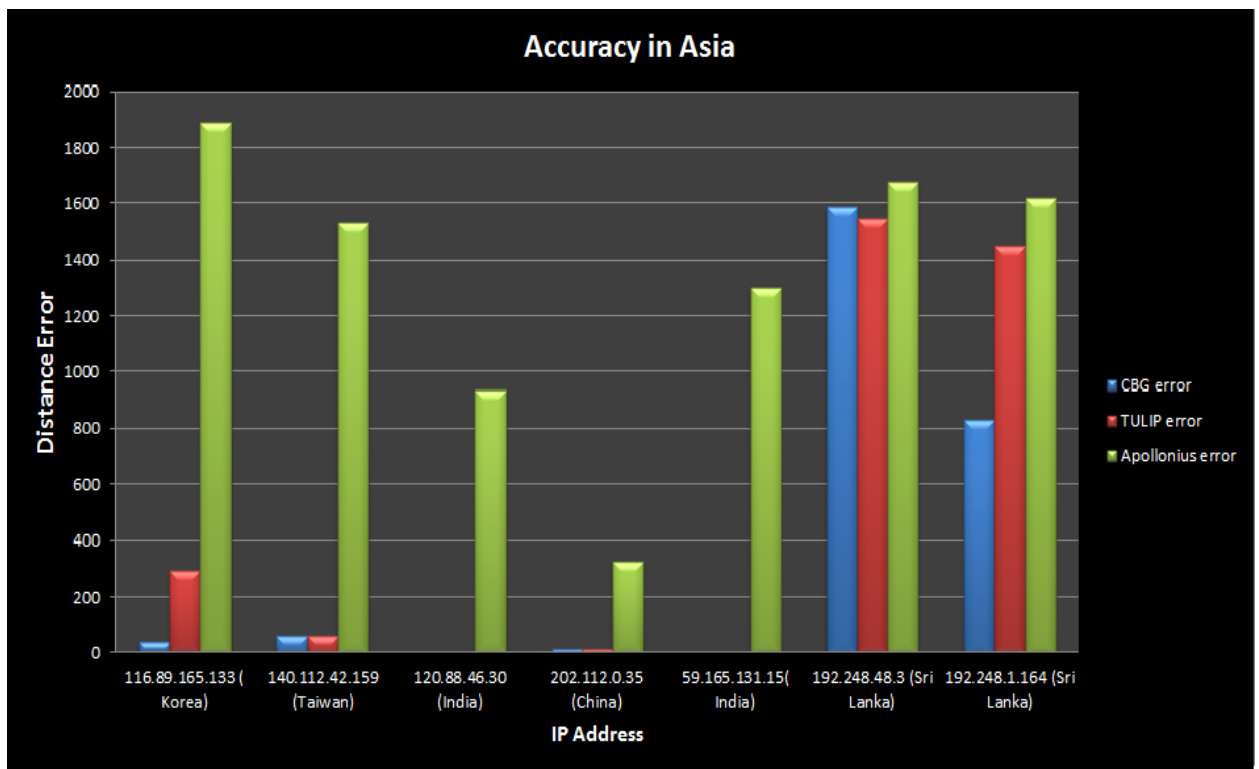


Figure 28: Accuracy in Asia graph

10.2.2 Accuracy of Europe

Table 7: Accuracy in Europe

Country	Target	Lat/Lon	GeoIP	CBG	Trilateration	Apollonius
Andorra	194.158.78.228	Latitude	42.5	53.855	51.5722	50.744
		Longitude	1.5167	-1.3099	-1.3099	-17.1345
Belarus	74.125.53.121	Latitude	37.4192	47.459	49.2119	40.063
		Longitude	-122.0574	-123.19	-123.103	-118.509
Belgium	193.190.198.39	Latitude	50.8333	51.622	51.5722	52.147
		Longitude	4.3333	-1.3099	-1.3099	0.9666
Denmark	130.225.212.5	Latitude	55.7	51.384	51.5722	57.183

		Longitude	12.5167	0.88071	-1.3099	0.6766
France	132.227.74.51	Latitude	48.867	49.868	51.5722	52.896
		Longitude	2.3333	2.2246	-1.3099	0.880
Germany	140.181.64.223	Latitude	49.8706	47.798	46.2325	49.490
		Longitude	8.6494	4.4836	6.0459	8.655

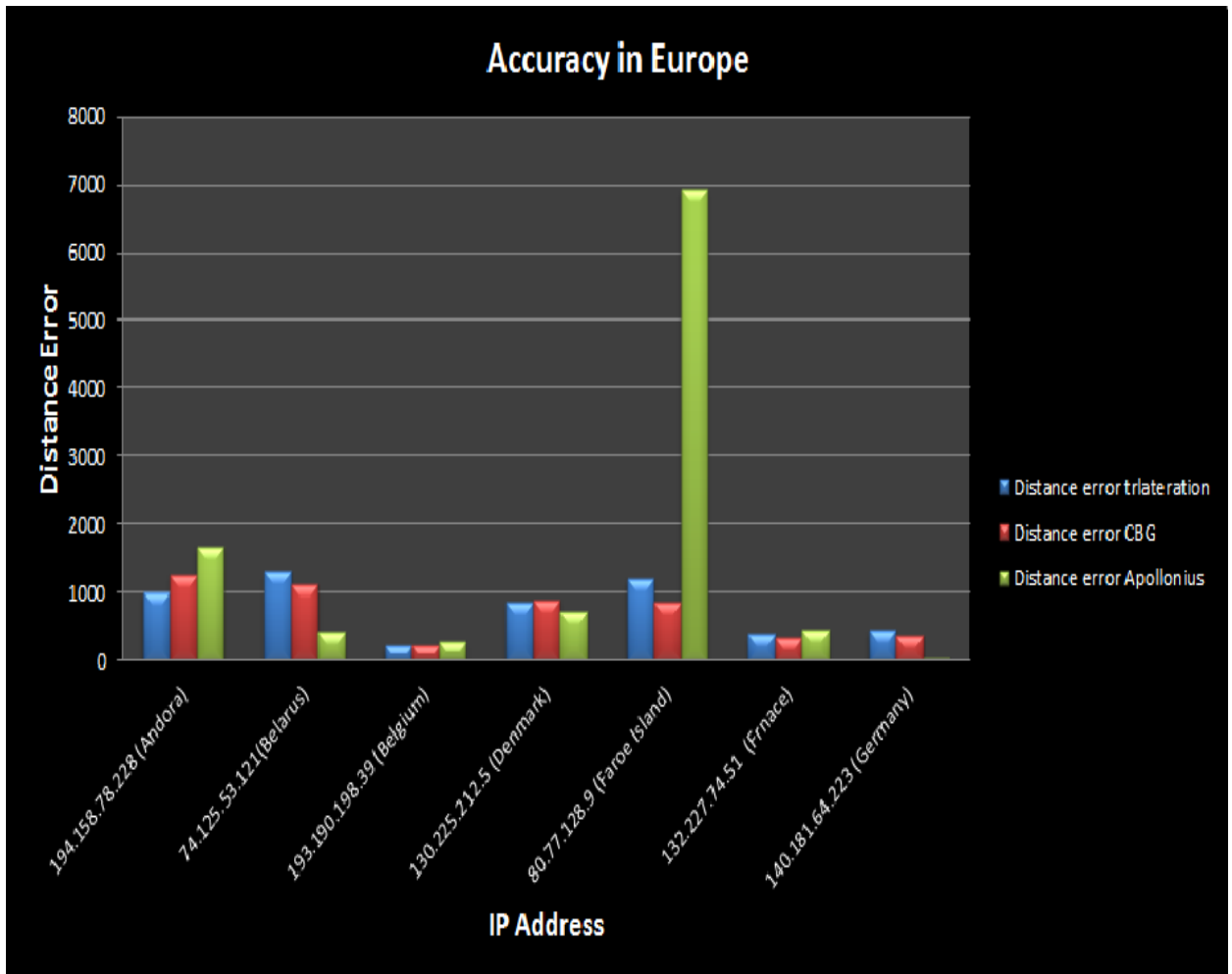


Figure 29: Accuracy in Europe graph

10.2.3 Accuracy in Africa

Table 8: Accuracy in Africa

Country	Target	Lat/Lon	GeoIP	CBG	Trilateration	Apollonius
Morocco	81.192.184.85	Latitude	34.0528	53.343	53.3431	39.700
		Longitude	-4.9828	-2.6407	-2.6407	-32.314
Algeria	193.194.64.71	Latitude	36.7631	36.9	36.9	49.697
		Longitude	3.0506	2.9	2.9	16.284
Botswana	168.167.168.34	Latitude	-24.6464	-42.041	-33.9658	-89.824
		Longitude	25.9119	18.168	18.1684	-948.25
Burkina Faso	206.82.130.77	Latitude	12.3702	12.666	12.377	33.687
		Longitude	-1.5247	-1.53	-1.53	10.811
Cape Verde	195.8.3.90	Latitude	14.9167	53.343	51.5722	55.759
		Longitude	-23.5168	-2.6407	-1.3099	-35.900
Ethiopia	213.55.76.98	Latitude	8	51.572	51.5722	54.080
		Longitude	38	-1.3099	-1.3099	-2.844

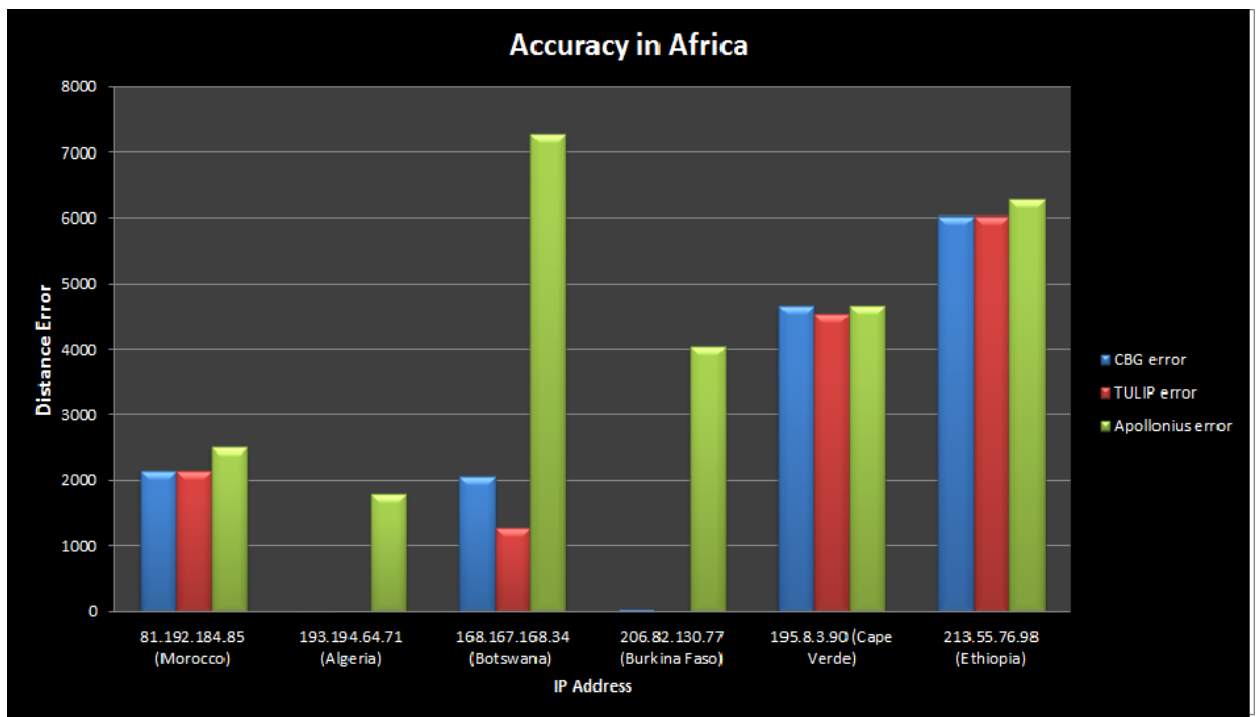


Figure 30: Accuracy in Africa graph

10.2.4 Accuracy in North America

Table 9: Accuracy in North America

Country	Target	Lat/Lon	GeoIP	CBG	Trilateration	Apollonius
United States	192.42.83.251	Latitude	41.8471	45.544	44.98	42.7935
		Longitude	-87.6248	-	-93.2638	-90.063
Laval	64.15.128.187	Latitude	45.6	43.692	45.42	41.934
		Longitude	-73.7333	-	-75.7	-73.169
Santa Clara	198.175.112.105	Latitude	37.3558	37.539	37.4177	39.8313
		Longitude	-	-122.2	-122.203	-128.66
Stanford	171.64.7.115	Latitude	37.4178	37.418	37.4177	40.920
		Longitude	-122.172	-122.2	-122.203	-122.478

MIT(USA)	18.7.22.69	Latitude	42.3646	42.343	42.3824	40.451
		Longitude	-71.1028	-	-71.0997	-73.24
North America	128.197.26.35	Latitude	42.3451	42.384	42.3824	40.604
		Longitude	-71.0993	-71.1	-71.0997	-73.308
United States	129.79.78.192	Latitude	38.739	41 .217	44.98	38.708
		Longitude	-87.157	-	-93.2638	-87.16

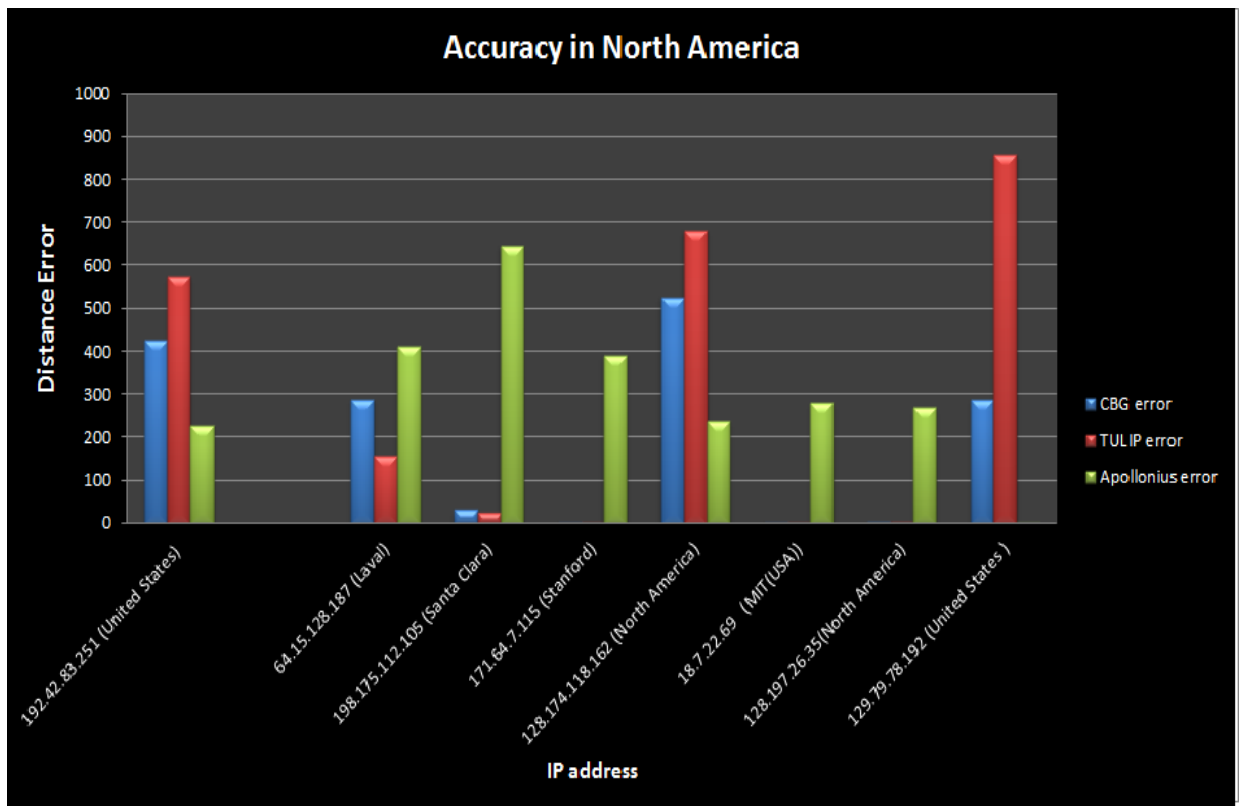


Figure 31: Accuracy in North America

10.2.5 Average distance error comparison w.r.t continents



Figure 32: Average distance error w.r.t to continents

10.2.6 Average distance error comparison w.r.t countries

Table 10: Average distance error comparison w.r.t to world wide

Country	Target	Lat/Lon	GeoIP	CBG	Trilateration	Apollonius
France	132.227.74.51	Latitude	48.867	49.868	51.5722	52.896
		Longitude	2.3333	2.2246	-1.3099	0.880
Taiwan	140.112.42.159	Latitude	25.0392	24.784	24.7812	21.507
		Longitude	121.525	120.99	120.993	136.026
Italy	143.225.229.236	Latitude	46.578	46.578	45.6486	40.066
		Longitude	13.78	13.78	13.78	0.950

United States	192.42.83.251	Latitude	41.8471	45.544	44.98	42.7935
		Longitude	-87.6248	-86.263	-93.2638	-90.063
Korea	116.89.165.133	Latitude	24.788	25.051	22.3361	35.331
		Longitude	115.33	115.54	114.264	130.803
United States	131.247.2.245	Latitude	28.0631	41.691	42.3824	19.963
		Longitude	-82.4128	-87.063	-71.0997	-101.415
Japan	133.15.59.2	Latitude	36	24.781	33.75	29.551
		Longitude	138	120.99	73.165	119.113
Germany	195.37.16.121	Latitude	48.5833	49.212	50.6833	50.009
		Longitude	13.4832	3.4329	10.9	8.400
Santa Clara	198.175.112.105	Latitude	37.3558	37.539	37.4177	39.8313
		Longitude	-121.9537	-122.2	-122.203	-128.66
Guatemala	168.234.74.100	Latitude	15.5	29.763	32.7781	11.195
		Longitude	-90.25	-95.383	-96.7982	-74.244

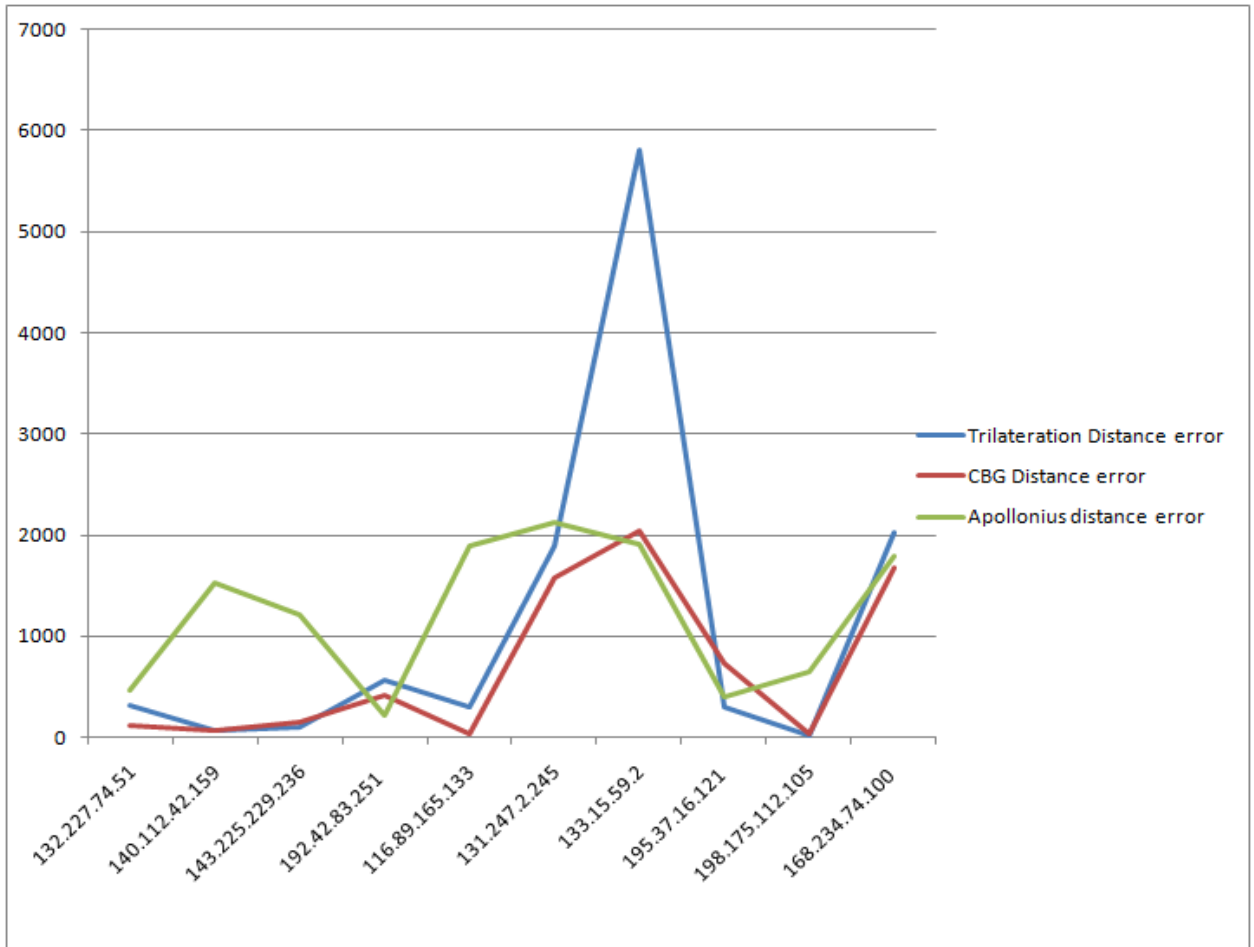


Figure 33: Average distance comparison world wide

We compared our results with the GeoIP because actual data is already stored in GeoIP database (actual latitude, longitude, city name, country name, continent name and distance). When we say that CBG is more accurate than Trilateration or Apollonius, it is expected that its results are much closer to GeoIP as compare to results of Trilateration and Apollonius. It's also seen that with less number of landmarks CBG will perform well as compared to previous techniques.

11 DISCUSSION

The results that we have seen from this project are such that more accurate results are achieved for targets in regions like North America and South Asia due to good network infrastructure and due to more monitoring and remote nodes as compared to any other areas. Similarly the results in Africa and Australia are not much accurate because of less number of nodes and network infrastructure. Currently CBG is running on two constraints cbg2 (Speed of light) (15.7) and cbg_soi (Speed of internet). Cbg_soi (15.8) results are more accurate than cbg2. On the other hand bestline constraint provides more accurate results but it is not integrated yet due to insufficient data provided by TULIP database. It requires data from an additional data source.

12 CONCLUSION

We already know that Internet has become a collection of resources meant to appeal to a large general audience. Although this multitude of information has been a great boon, it also has diluted the importance of geographically localized information. Offering the ability for Internet users to garner information based on geographic location can decrease Search times and increase visibility of local establishments. Geolocation by IP address is the technique of determining a user's geographic latitude, longitude and, by Inference, city, region and nation by comparing the user's public Internet IP address with known locations of other electronically neighboring servers and routers.

So In this document, we described the CBG-TULIP integration; both are measurement-based method to estimate the geographic location of Internet hosts. Based on delay measurements, CBG use Trilateration or Multilateration to infer a location estimate for a given target host. The experimental results show that CBG perform well as compared to other existing techniques because its accuracy is almost 70% which is greater than the all other existing technique. Our results are based on measurements taken in well connected, geographically contiguous networks. To some extent our work takes advantage of the fact that network connectivity has improved dramatically in the last decade, and that the relationship between network delay and geographic distance is strong in these regions Thus one must be cautious before extrapolating our results to arbitrary network regions. CBG establishes a dynamic relationship between network delay and geographic distance. This is done in a distributed and self-calibrating fashion among the adopted landmarks using the constraint SOI (cbg_soi) we can also use other constraint like speed of light (cbg2.m) and Bestline (bestline.m) .In addition to some expected sources of distortion in this relationship, such as queuing delay and the absence of great-circle paths, our results

point out other sources as well. The presence of shared paths hides the location of the target host behind a single point, also leading to inaccurate estimates. But basic purpose of this project is to integrate the CBG with other existing algorithm like Apollonius and GeoIP so we have multiple ways to view the results on Map and compare these results with each other and find the most accurate result.

13 FUTURE WORK

We also know that no software is perfect .So still there are also some issues are existing in this architecture which can be solve by using these techniques.

13.1 USING BESTLINE APPROACH

As we now CBG is running on three constraints:

- Speed of Light
- Speed of Internet
- Bestline approach

Currently we have deployed first two constraints but regarding accuracy the third constraint is more accurate. Basically TULIP code provides us the data only by which we can measure first two constraints. For third constraint CBG will requires some extra information which is not currently available from TULIP code.

The input of CBG with respect to different constraint is such that for first two constraints it only needs data of Landmarks i.e. Lat/Lon and their RTTs with the specified target. For bestline approach we not only required this data but also requires data for all those targets which are pinged by these Landmarks for example their RTTs from that one landmark which is in group of landmarks who ping the target whose location is to be determined.

For extracting this extra information we require to have connection to a database that stores RTTs of different landmarks to different targets. When this Integration code runs for estimating a target then at run time it connects to this database and fetches data for those landmarks which are called by reflector.cgi to ping the given target. Most probably this database is of PingER which stores data hourly for analysis purposes.

13.2 LANDMARK TIERING ANALYSIS

Basically it will help selecting only those landmarks that are near to the testing target. Another approach towards effective landmark selection could be tiering approach. In this approach first we probe a target host with a few selected level zero Landmarks once the tier has been Landmarks of that tier to increase accuracy. Following is a case study for the feasibility of tiering approach for North American region. The purpose of this study is to investigate the effectiveness of tiering for TULIP (i.e. we have a set of primary landmarks tier0 which will narrow down the target location to being in a particular region and then a denser set of secondary tier1 landmarks in the discovered region that can be used to get more accurate results). The use of tiering should enable us to reduce the network traffic (number of landmarks pinging a target) while retaining the accuracy of using all landmarks.

14 REFERENCES

- [1] <http://pinger.seecs.edu.pk/tutorial/tutorial.html#mechanism>
- [2] ICFA SCIC Network Monitoring Report Prepared by the ICFA SCIC Monitoring Working Group published January 2010.
- [3] A Hands-On Guide to Relational Database Design By Michael J. Hernandez Chapter 2 Design Objectives
- [5] <https://confluence.slac.stanford.edu/display/IEPM/Pinger+NODEDETAILS>
- [6] <http://pinger.seecs.edu.pk/tutorial/tutorial.html#gather>
- [7] <https://confluence.slac.stanford.edu/dosearchsite.action?queryString=tulip>
(TULIP)
- [8] <https://confluence.slac.stanford.edu/display/IEPM/TULIP+Map+%28Landmarks%29>
- [9] <http://en.wikipedia.org/wiki/Geolocation>
- [10] <http://portal.acm.org/citation.cfm?id=1028828>
- [11] <https://confluence.slac.stanford.edu/display/IEPM/TULIP+Web+Based+Visualization>

15 APPENDICIES

15.1 CHANGES IN NODE.PL

```
my $dbh = DBI->connect("DBI:mysql:archive3", 'root', 'chishti') || die "Could not connect to database: $DBI::errstr";
if($count==0){
my $query1t = "truncate node_details";
my $query_handle1 = $dbh->prepare($query1t);
$query_handle1->execute();
$count++;
}
my $query1 = "INSERT INTO node_details VALUES (null,'$row->{NODENAME}', '$row->{IPADDRESS}', '$row->{SITENAME}',
'$row->{NICKNAME}', '$row->{COUNTRY}', '$row->{CONTINENT}',
'$row->{LAT}', '$row->{LONG}', '$row->{PROJECTTYPE}', '$row->{PINGSERVER}', '$row->{TRACESERVER}',
'$row->{DATASERVER}', '$row->{URL}', '$row->{GMT}')";
my $query_handle = $dbh->prepare($query1);
$query_handle->execute();
```

15.2 CHANGE IN GETDATA.PL

```
my $dbh = DBI->connect("DBI:mysql:archive3", 'root', 'chishti') || die "Could not connect to database: $DBI::errstr";
my $query2 = "select nodename,dataserver from node_details where (projecttype='M' || projecttype='MB' || projecttype='MD' ||
projecttype='MWB' || projecttype='DM') and country='Pakistan'";
my $query_handle = $dbh->prepare($query2);
$query_handle->execute();
$query_handle->bind_columns(undef, \ $nd, \ $ds);
my @nodename;
my @data_servers;
my $count=0;
while($query_handle->fetch()){
@nodename[$count]=$nd;
@data_servers1[$count]=$ds;
$count++;}
$count1=0;
***** Insert values*****
my $query1 = "INSERT INTO pingdata VALUES ($id2, $id3, $vars[4], '$vars[5]', $vars[6], $vars[7], $vars[8], $vars[9], $vars[10],
$vars[11], $vars[12], $vars[13], $vars[14], $vars[15], $vars[16], $vars[17], $vars[18], $vars[19], $vars[20], $vars[21], $vars[22],
$vars[23], $vars[24], $vars[25], $vars[26], $vars[27], $vars[28], $vars[29], $vars[30], $date)";
my $query_handle = $dbh->prepare($query1);
$query_handle->execute();
```

There are also changes to handle data. That cannot be mentioned except writing the whole script here.

15.3 CHANGES IN ANALYZE HOURLY

```
$query1s = "SELECT count(*) from pingdata where date='$process_year-$process_mon-$process_mday' and
monnode_id='$idmn'";
$query_handle = $dbh->prepare($query1s);
$query_handle->execute();
```



```

$query_handle->bind_columns(undef, \%ct);
$query_handle->fetch();
if(\$ct==0)
{print "\nNo data for date $process_year-$process_mon-$process_mday for monitoring site $monitoring_site exists.\n";
$countmns++;
print "Number of no data monitoring sites are $countmns";
next;
}
}
$mon_site++;
*****Read Data*****
my $count1=0;
$query2s = "SELECT a.nodename,a.sitename,a.ipaddress,a.projecttype,a.nickname,a.lat,a.lon,
b.nodename,b.sitename,b.ipaddress,b.projecttype,b.nickname,b.lat,b.lon,
c.packet_size,c.time,c.sent,c.recv,c.min_rtt,c.avg_rtt,c.max_rtt,c.seq1,c.seq2,c.seq3,c.seq4,
c.seq5,c.seq6,c.seq7,c.seq8,c.seq9,c.seq10,c.rtt1,c.rtt2,c.rtt3,c.rtt4,c.rtt5,
c.rtt6,c.rtt7,c.rtt8,c.rtt9,c.rtt10
FROM node_details a,node_detailsb,pingdata c
WHERE c.date='$process_year-$process_mon-$process_mday' and
c.monnode_id=a.id and
c.remnode_id=b.id and c.monnode_id='$idmn';
$query_handle = $dbh->prepare($query2s);
$query_handle->execute();
$query_handle->bind_columns(undef, \%mon_name, \%mon_sitename, \%mon_ip, \%mon_proj, \%mon_nick, \%mon_lat,
\%mon_lon, \%rem_name, \%rem_sitename, \%rem_ip, \%rem_proj, \%rem_nick, \%rem_lat, \%rem_lon, \%pk_size, \%time, \%sent,
\%recv, \%min_rtt, \%avg_rtt, \%max_rtt, \%seq1, \%seq2, \%seq3, \%seq4, \%seq5, \%seq6, \%seq7, \%seq8, \%seq9, \%seq10, \%rtt1,
\%rtt2, \%rtt3, \%rtt4, \%rtt5, \%rtt6, \%rtt7, \%rtt8, \%rtt9, \%rtt10);
LINE:
while($query_handle->fetch()) {
$line++;
my $from=$mon_name;
if($from =~ /\{1,3\}\.\{1,3\}\.\{1,3\}\.\{1,3\}/) {
if(($debug>0) && ($line<2)) {
print STDERR " Warning mon host for $monitoring_site has only IP address=$from\n";
}
}
*****Store in analysis table*****
my $query1 = "INSERT INTO analysis VALUES ('$monnn', '$remmm', '$size', '$by', '$clp', '$min_rtt', '$pur', '$mpl', '$pup',
'$ipdv', '$zplf', '$pk_loss', '$alphaa', '$dp', '$iqr', '$avg_rtt', '$thr', '$oop', '$moss', '$yearc', '$monc', '$mday', '$i)";
my $query_handle = $dbh->prepare($query1);
$query_handle->execute();

```

15.4 CHANGES IN ANALYZE-DAILY.PL

```

*****Reading data*****:
$query1s = "SELECT a.mon,a.rem,a.hour,a.$metric_select,a.mon,a.rem from analysis a where a.year='$year' and
a.month='$mon' and a.day='$mday' and a.pksize='$size' and a.byclass='$by' and hour is not null order by a.mon,a.rem,a.hour";
$query_handle = $dbh->prepare($query1s);
$query_handle->execute();
#$query_handle->bind_columns(undef, \%monh, \%remh, \%hh, \%mt, \%monht, \%remht, \%monst, \%remst);
$query_handle->bind_columns(undef, \%monh, \%remh, \%hh, \%mt, \%monht, \%remht);
LINE:
while($query_handle->fetch()) {

#print "\n $monh $remh ----- $hh ---- $mt ----- $monht $remht \n";
$ele[$hh]=$mt;
if($hheq '23')
{
$line="$monh $remh $ele[0] $ele[1] $ele[2] $ele[3] $ele[4] $ele[5] $ele[6] $ele[7] $ele[8] $ele[9] $ele[10] $ele[11] $ele[12]
$ele[13] $ele[14] $ele[15] $ele[16] $ele[17] $ele[18] $ele[19] $ele[20] $ele[21] $ele[22] $ele[23] $monht $remht";
}
}

```

```

*****Storing data*****
my $query1 = "INSERT INTO analysis VALUES ('$monnn', '$remmm', '$size', '$by', '$clp', '$min_rtt', '$pur', '$mpl', '$pup',
'$ipdv', '$zplf', '$pk_loss', '$alphaa', '$dp', '$iqr', '$avg_rtt', '$thr', '$oop', '$moss', '$year', '$mon', '$mday', null)";
my $query_handle = $dbh->prepare($query1);
$query_handle->execute();
print "\n\nOne row Inserted\n\n";

```

15.5 CHANGES IN ANALYZE-MONTHLY.PL

```

$query1s = "SELECT a.mon,a.rem,a.day,a.$metric_select,a.mon,a.rem from analysis a where a.year='$year' and
a.month='$mon' and a.hour is null and a.day is not null and a.pksize='$size' and a.byclass='$by' order by a.mon,a.rem,a.day";
$query_handle = $dbh->prepare($query1s);
$query_handle->execute();
#$query_handle->bind_columns(undef, \ $monh, \ $remh, \ $hh, \ $mt, \ $monht, \ $remht, \ $monst, \ $remst);
$query_handle->bind_columns(undef, \ $monh, \ $remh, \ $hh, \ $mt, \ $monht, \ $remht);
LINE:
while($query_handle->fetch()) {
print "\n $monh $remh ----- $hh ---- $mt ----- $monht $remht \n";
$ele[$hh-1]=$mt;
if($hheq $days)
{
if($hh==31)
{
$line="$monh $remh $ele[0] $ele[1] $ele[2] $ele[3] $ele[4] $ele[5] $ele[6] $ele[7] $ele[8] $ele[9] $ele[10] $ele[11] $ele[12]
$ele[13] $ele[14] $ele[15] $ele[16] $ele[17] $ele[18] $ele[19] $ele[20] $ele[21] $ele[22] $ele[23] $ele[24] $ele[25] $ele[26]
$ele[27] $ele[28] $ele[29] $ele[30] $monht $remht";
}

if($hh==30)
{
$line="$monh $remh $ele[0] $ele[1] $ele[2] $ele[3] $ele[4] $ele[5] $ele[6] $ele[7] $ele[8] $ele[9] $ele[10] $ele[11] $ele[12]
$ele[13] $ele[14] $ele[15] $ele[16] $ele[17] $ele[18] $ele[19] $ele[20] $ele[21] $ele[22] $ele[23] $ele[24] $ele[25] $ele[26]
$ele[27] $ele[28] $ele[29] $monht $remht";
}

if($hh==28)
{
$line="$monh $remh $ele[0] $ele[1] $ele[2] $ele[3] $ele[4] $ele[5] $ele[6] $ele[7] $ele[8] $ele[9] $ele[10] $ele[11] $ele[12]
$ele[13] $ele[14] $ele[15] $ele[16] $ele[17] $ele[18] $ele[19] $ele[20] $ele[21] $ele[22] $ele[23] $ele[24] $ele[25] $ele[26]
$ele[27] $monht $remht";
}

if($hh==29)
{
$line="$monh $remh $ele[0] $ele[1] $ele[2] $ele[3] $ele[4] $ele[5] $ele[6] $ele[7] $ele[8] $ele[9] $ele[10] $ele[11] $ele[12]
$ele[13] $ele[14] $ele[15] $ele[16] $ele[17] $ele[18] $ele[19] $ele[20] $ele[21] $ele[22] $ele[23] $ele[24] $ele[25] $ele[26]
$ele[27] $ele[28] $monht $remht";
}

print "\n $line \n";
*****Store in analysis Table*****
my $query1 = "INSERT INTO analysis VALUES ('$monnn', '$remmm', '$size', '$by', '$clp', '$min_rtt', '$pur', '$mpl', '$pup',
'$ipdv', '$zplf', '$pk_loss', '$alphaa', '$dp', '$iqr', '$avg_rtt', '$thr', '$oop', '$moss', '$year', '$mon', null, null)";
my $query_handle = $dbh->prepare($query1);
$query_handle->execute();
print "\n\nOne row Inserted\n\n";

```

15.6 CHANGES IN PINGTABLE.PL

```

if($FORM{'tick'}=~~/hourly/) {
$topline="0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 ";#First line contains the heading
$vartt=23;

```

```

my $query21 = "select count(*) from analysis where year='$FORM{year}' and month='$FORM{month}' and
day='$FORM{day}'
and pksize='$FORM{size}' and byclass='$FORM{by}'";
my $query_handle = $dbh->prepare($query21);
$query_handle->execute();
$query_handle->bind_columns(undef, \ $count_hourly);
$query_handle->fetch();
$count_interval=$count_hourly;
$query1hds = "SELECT a.mon,a.rem,a.hour,a.hour,a.$metric_select,a.mon,a.rem from analysis a where
a.year='$FORM{year}' and a.month='$FORM{month}' and a.day='$FORM{day}' and a.pksize='$FORM{size}' and
a.byclass='$FORM{by}' order by a.mon,a.rem,a.hour";
#*****Display Daily Data*****
elsif($FORM{tick}'~/daily/) {
undef $FORM{day};
my $mon=$FORM{month};
if( isLeapYear($year) eq '1'){
    $days = $days_in_month_leapYear[$mon-1];
}
else {
    $days=$days_in_month[$mon-1];
}
}
$varrt = $days;
for $day (1..$days) {
    $year=$FORM{year};
    $mon=sprintf "%2.2d", $mon;
    $mday=sprintf "%2.2d", $day; #Create header label of form yymmdd, e.g. 07May01 (= May 1st 2007)
    $title[$day]=substr(sprintf("%2.2d", $year),2,2).$months[$mon-1].$mday;
    $topline = $topline."$title[$day] ";
    # $topline = "$title[$day] ". $topline;
}
my $query21 = "select count(*) from analysis where year='$FORM{year}' and month='$FORM{month}' and
pksize='$FORM{size}' and byclass='$FORM{by}' and hour is null";
my $query_handle = $dbh->prepare($query21);
$query_handle->execute();
$query_handle->bind_columns(undef, \ $count_daily);
$query_handle->fetch();
$count_interval=$count_daily;
$query1hds = "SELECT a.mon,a.rem,a.hour,a.day,a.$metric_select,a.mon,a.rem from analysis a where a.year='$FORM{year}'
and a.month='$FORM{month}' and a.hour is null and a.day is not null and a.pksize='$FORM{size}' and
a.byclass='$FORM{by}' order by a.mon,a.rem,a.day";
}
#*****Display last 60 days data*****
elsif($FORM{tick}'~/last60days/) {
undef $FORM{day}; undef $FORM{month};
# $config{DATAFILE} = "$PATH{REPORT_DIR}/$FORM{dataset}/$FORM{file}/$FORM{file}-$FORM{size}-$FORM{by}-
60days.txt.gz";
$days=60;
my $seconds_ago=($days)*24*60*60;
my $start_time=time-$seconds_ago;
my @yyear;
my @mmonth;
my @dday;
my $ccountty=0;
my $ccounttm=0;
my $ccounttd=0;
$varrt = $days;
for $day (1..$days) {
    $unix_time=$start_time+($day-1)*86400;
    ($sec,$min,$hour,$mday,$mon,$year,$wday,$yday,$isdst) = gmtime($unix_time);

```

```

$year+=1900;
if($ccounty>0)
{
if($year==$year[$ccounty-1]) { }
else {
$year[$ccounty]=$year;
$ccounty++;
}}else
{$year[$ccounty]=$year;
$ccounty++;
}
$mon=sprintf "%2.2d", $mon+1;
if($ccounttm>0)
{
if($mon==$mmonth[$ccounttm-1]) { }
else
{
$mmonth[$ccounttm]=$mon;
$ccounttm++;}
}
else
{
$mmonth[$ccounttm]=$mon;
$ccounttm++;
}
}
$mday=sprintf "%2.2d", $mday;
if($ccounttd>0)
{
if($mday==$dday[$ccounttd-1]) { }
else
{
$dday[$ccounttd]=$mday;
$ccounttd++;}
#Create header label of form yymmdd, e.g. 07May01 (= May 1st 2007)
$title[$day]=substr(sprintf("%2.2d",$year),2,2).$months[$mon-1].$mday;
#$stipline =$stipline."$title[$day] ";
$stipline ="$title[$day] ".$stipline;
}
if($#mmonth+1==2)
{
my @md1;
my @md2;
my $ccutt=0;
my $sct1=0;
my $asd=0;
for( $t=0 ; $t<60 ; $t++ )
{

$asd++;

if ($dday[$t]==1)
{
if($asd!=1)
{
$sct1++;
}
}
$ccutt=0;
}
if($sct1==0)

```

```

{
$md1[$scutt]=$dday[$t];
$scutt++;
}
if($sct1==1)
{
$md2[$scutt]=$dday[$t];
$scutt++;
}
$query1hds = "SELECT mon,rem,month,day,$metric_select,mon,rem from analysis where year=$year and
((month=$mmonth[0] and day >=$md1[0]) or (month=$mmonth[1] and day<=$md2[$#md2])) and hour is null and day is not null
and pksize='$FORM{size}' and byclass='$FORM{by}' order by mon,rem,month,day,year";
}
if($#mmonth+1==3)
{
my @md1;
my @md2;
my @md3;
my $scutt=0;
my $sct1=0;
for( $t=0 ; $t<60 ; $t++ )
{
if ($dday[$t]==1)
{
$sct1++;
$scutt=0;
}
if($sct1==0)
{
$md1[$scutt]=$dday[$t];
$scutt++;
}
if($sct1==1)
{
$md2[$scutt]=$dday[$t];
$scutt++;
}
if($sct1==1)
{
$md2[$scutt]=$dday[$t];
$scutt++;
}
if($sct1==2)
{
$md3[$scutt]=$dday[$t];
$scutt++;
}
}
}
$cutmon=$mmonth[2];
$cutday=$md3[$#md3];
$query1hds = "SELECT mon,rem,month,day,$metric_select,mon,rem from analysis where year=$year and (month=$mmonth[1]
or (month=$mmonth[0] and day >=$md1[0]) or (month=$mmonth[2] and day<=$md3[$#md3])) and hour is null and day is not
null and pksize='$FORM{size}' and byclass='$FORM{by}' order by mon,rem,month,day,year";

}
my $query21 = "select count(*) from analysis where year=$year and month=$mon and pksize='$FORM{size}' and
byclass='$FORM{by}' and hour is null";
my $query_handle = $dbh->prepare($query21);
$query_handle->execute();
$query_handle->bind_columns(undef, \ $count_daily);
$query_handle->fetch();

```

```

$count_interval=$count_daily;
}
#*****Last 120 Days*****
elseif($FORM{'tick'}=~//last120days/) {
undef $FORM{'day'}; undef $FORM{'month'};
# $config{'DATAFILE'} = "$PATH{'REPORT_DIR'}/$FORM{'dataset'}/$FORM{'file'}/$FORM{'file'}-$FORM{'size'}-$FORM{'by'}-
120days.txt.gz";
$days=120;
my $seconds_ago=($days)*24*60*60;
my $start_time=time-$seconds_ago;
my @yyear;
my @mmonth;
my @dday;
my $ccounty=0;
my $ccounttm=0;
my $ccounttd=0;
$vartt = $days;

my $hahaha=0;
for $day (1..$days) {
    $unix_time=$start_time+($day-1)*86400;
    ($sec,$min,$hour,$mday,$mon,$year,$wday,$yday,$isdst) = gmtime($unix_time);
    $year+=1900;
    if($ccounty>0)
    {
    if($year==$yyear[$ccounty-1]) { }
    else {
    $yyear[$ccounty]=$year;
    $ccounty++;
    }
    }
    else
    {
    $yyear[$ccounty]=$year;
    $ccounty++;
    }
    $mon=sprintf "%2.2d", $mon+1;
    if($ccounttm>0)
    {
    if($mon==$mmonth[$ccounttm-1]) { }
    else
    {
    $mmonth[$ccounttm]=$mon;
    $ccounttm++;
    }
    }
    else
    {
    $mmonth[$ccounttm]=$mon;
    $ccounttm++;
    }
    $mday=sprintf "%2.2d", $mday;
    if($ccounttd>0)
    {
    if($mday==$dday[$ccounttd-1]) { }
    else
    {
    $dday[$ccounttd]=$mday;

```

```

$counttd++;
}
}else
{
$dday[$counttd]=$mday;
$counttd++;
}
#Create header label of form yymmdd, e.g. 07May01 (= May 1st 2007)
$title[$day]=substr(sprintf("%2.2d",$year),2,2).$months[$mon-1].$mday;
#$stpline =$stpline."$title[$day] ";
$stpline ="$title[$day] ".$stpline;
}
if($#mmonth+1==4)
{
my @md1;
my @md2;
my @md3;
my @md4;
my $cctt=0;
my $sct1=0;
for( $t=0 ; $t<120 ; $t++ )
{

if ($dday[$t]==1)
{
$sct1++;
$cctt=0;
}
if($sct1==0)
{
$md1[$cctt]=$dday[$t];
$cctt++;
}
if($sct1==1)
{
$md2[$cctt]=$dday[$t];
$cctt++;
}
if($sct1==2)
{
$md3[$cctt]=$dday[$t];
$cctt++;
}
if($sct1==3)
{
$md4[$cctt]=$dday[$t];
$cctt++;
}
}
$cutmon=$mmonth[3];
$cutday=$md4[$#md4];
$query1hds = "SELECT mon,rem,month,day,$metric_select,mon,rem from analysis where year=$year and
((month=$mmonth[0] and day >=$md1[0]) or month=$mmonth[1] or month=$mmonth[2] or (month=$mmonth[3] and
day<=$md4[$#md4])) and hour is null and day is not null and pksize=$FORM{'size'} and byclass=$FORM{'by'} order by
mon,rem,month,day,year";
}
if($#mmonth+1==5)
{
#print "\n Five months \n";
my @md1;

```

```

my @md2;
my @md3;
my @md4;
my @md5;
my $cctt=0;
my $cct1=0;
for( $t=0 ; $t<120 ; $t++ )
{
if ($dday[$t]==1)
{
$cct1++;
$cctt=0;
}
if($cct1==0)
{
$md1[$cctt]=$dday[$t];
$cctt++;
}
if($cct1==1)
{
$md2[$cctt]=$dday[$t];
$cctt++;
}if($cct1==2)
{
$md3[$cctt]=$dday[$t];
$cctt++;
}
if($cct1==3)
{
$md4[$cctt]=$dday[$t];
$cctt++;
}
if($cct1==4)
{
$md5[$cctt]=$dday[$t];
$cctt++;
}
}
$cutmon=$mmonth[4];
$cutday=$md5[$#md5];
$query1hds = "SELECT mon,rem,month,day,$metric_select,mon,rem from analysis where year=$year and
((month=$mmonth[0] and day >=$md1[0]) or month=$mmonth[1] or month=$mmonth[2] or month=$mmonth[3] or
(month=$mmonth[4] and day<=$md5[$#md5])) and hour is null and day is not null and pksize='$FORM{'size'}' and
byclass='$FORM{'by'}' order by mon,rem,month,day,year";
}
my $query21 = "select count(*) from analysis where year=$year and month=$mon and pksize='$FORM{'size'}' and
byclass='$FORM{'by'}' and hour is null";
my $query_handle = $dbh->prepare($query21);
$query_handle->execute();
$query_handle->bind_columns(undef, \ $count_daily);
$query_handle->fetch();
$count_interval=$count_daily;
#*****Monthly Data*****
else {
undef $FORM{'day'}; undef $FORM{'month'};
# $config{'DATAFILE'} = "$PATH{'REPORT_DIR'}/$FORM{'dataset'}/$FORM{'file'}/$FORM{'file'}-$FORM{'size'}-
$FORM{'by'}.txt.gz";
$varrt = 24;
my ($sec,$min,$hour,$mday,$mon,$year,$wday,$yday,$isdst) = gmtime(time);

```



```

my $start_time=timegm(0,0,0,15,$mon,$year-2);
my @yyear;
my @mmonth;
#my @dday;
my $ccounty=0;
my $ccounttm=0;
#my $ccounttd=0;
for my $month (1..24) {
    $unix_time=$start_time+$month*86400*30;
    ($sec,$min,$hour,$mday,$mon,$year,$wday,$yday,$isdst) = gmtime($unix_time);
    $year+=1900;
    if($ccounty>0)
    {
        if($year==$yyear[$ccounty-1]) { }
        else {
            $yyear[$ccounty]=$year;
            $ccounty++;
        }
        $mon=sprintf "%2.2d", $mon+1;
        if($ccounttm>0)
        {
            if($mon==$mmonth[$ccounttm-1]) { }
            else
            {
                $mmonth[$ccounttm]=$mon;
                $ccounttm++;
            }
        }
        else
        {
            $mmonth[$ccounttm]=$mon;

            $ccounttm++;
        }
        $title[$month]="$months[$mon-1]$year";

        #$stoline =$stoline."$title[$month] ";

        $stoline ="$title[$month] ". $stoline;
    }if($#yyear+1==2)
    {
        #print "\n Two years \n";
        my @md1;
        my @md2;
        #my @md3;
        my $ccut1=0;
        my $sct1=0;
        for( $t=0 ; $t<24 ; $t++ )
        {
            if ($mmonth[$t]==1)
            {
                $sct1++;
                $ccutt=0;
            }
        }
        if($sct1==0)
        {
            $md1[$ccutt]=$mmonth[$t];
            $ccutt++;
        }
    }

```

```

if($sct1==1)
{
$md2[$scutt]=$mmonth[$t];
$scutt++;
}$cutmon=$yyear[1];
$cutday=$md2[$#md2];
$query1hds = "SELECT mon,rem,year,month,$metric_select,mon,rem from analysis where (year=$yyear[0] and month
>=$md1[0]) or (year=$yyear[1] and month<=$md2[$#md2]) and hour is null and day is null and pksize='$FORM{'size}' and
byclass='$FORM{'by}' order by mon,rem,year,month";
}if($#yyear+1==3)
{
my @md1;
my @md2;
my @md3;
my $scutt=0;
my $sct1=0;
for( $t=0 ; $t<24 ; $t++ )
{
if ($mmonth[$t]==1)
{
$sct1++;
$scutt=0;
}
}
if($sct1==0)
{
$md1[$scutt]=$mmonth[$t];
$scutt++;
}
if($sct1==1)
{
$md2[$scutt]=$mmonth[$t];
$scutt++;
}if($sct1==1)
{
$md2[$scutt]=$mmonth[$t];
$scutt++;
}
}
if($sct1==2)
{
$md3[$scutt]=$mmonth[$t];
$scutt++;
}
}
$cutmon=$yyear[2];
$cutday=$md3[$#md3];
$query1hds = "SELECT mon,rem,year,month,$metric_select,mon,rem from analysis where year=$yyear[1] or (year=$yyear[0]
and month >=$md1[0]) or (year=$yyear[2] and month<=$md3[$#md3]) and hour is null and day is null and
pksize='$FORM{'size}' and byclass='$FORM{'by}' order by mon,rem,year,month";}
my $query21 = "select count(*) from analysis where year=$year and pksize='$FORM{'size}' and byclass='$FORM{'by}' and
hour is null and day is null";
my $query_handle = $dbh->prepare($query21);
$query_handle->execute();
$query_handle->bind_columns(undef, \$count_daily);
$query_handle->fetch();

$count_interval=$count_daily;
*****Select Monitoring Nodes*****
print "<br>From\n";
print "<select name=\"from\">\n";print "<option value=\"WORLD\">WORLD\n";

```

```

$query2s = "select distinct nickname from node_details where projecttype in('MB','M','MD','MWB','DM')";
$query_handle = $dbh->prepare($query2s);
$query_handle->execute();
$query_handle->bind_columns(undef, \%site);
while($query_handle->fetch()) {
print "<option value=\"$site\" ";
if ($site=~/^$FORM{from}$/) { print " selected"; }
print ">$site\n";}
$query2s = "select distinct country from node_details where projecttype in('MB','M','MD','MWB','DM')";
$query_handle = $dbh->prepare($query2s);
$query_handle->execute();
$query_handle->bind_columns(undef, \%site);
while($query_handle->fetch()) {
print "<option value=\"$site\" ";
if ($site=~/^$FORM{from}$/) { print " selected"; }
print ">$site\n";
}
}
$query2s = "select distinct continent from node_details where projecttype in('MB','M','MD','MWB','DM')";
$query_handle = $dbh->prepare($query2s);
$query_handle->execute();
$query_handle->bind_columns(undef, \%site);
while($query_handle->fetch()) {
print "<option value=\"$site\" ";
if ($site=~/^$FORM{from}$/) { print " selected"; }
print ">$site\n";
}
}
print "</select>\n";
#*****Select Remote Nodes*****
print "To\n";
print "<select name=\"to\">\n";
print "<option value=\"WORLD\">WORLD\n";
$query2s = "select distinct nickname from node_details where projecttype in('D','DA','ZNND','ZD','MD','ID','DI','DM')";
$query_handle = $dbh->prepare($query2s);
$query_handle->execute();
$query_handle->bind_columns(undef, \%site);
while($query_handle->fetch()) {
print "<option value=\"$site\" ";
if ($site=~/^$FORM{from}$/) { print " selected"; }
print ">$site\n";}
$query2s = "select distinct country from node_details where projecttype in('D','DA','ZNND','ZD','MD','ID','DI','DM')";
$query_handle = $dbh->prepare($query2s);
$query_handle->execute();
$query_handle->bind_columns(undef, \%site);
while($query_handle->fetch()) {
print "<option value=\"$site\" ";
if ($site=~/^$FORM{from}$/) { print " selected"; }
print ">$site\n"
}
}
$query2s = "select distinct continent from node_details where projecttype in('D','DA','ZNND','ZD','MD','ID','DI','DM')";
$query_handle = $dbh->prepare($query2s);
$query_handle->execute();
$query_handle->bind_columns(undef, \%site);
while($query_handle->fetch()) {
print "<option value=\"$site\" ";
if ($site=~/^$FORM{from}$/) { print " selected"; }
print ">$site\n";}
print "</select>\n";

```

15.7 CHANGES IN CBG2.M

```
%% run constraint-based geolocation using speed of light constraints
%% expects one file per target (see geolocate.m for format)
%% target.list should be a file listing all the target files
%% writes the results to estimates.cbg (see geolocateall.m for format)
%% wraps geolocateall, which creates its own output files with .cbg extension
```

```
function[]=cbg2(target)
tt = target;
extension = '.txt';
manifest = ['inputs/',tt, '.txt'];
resultfile = ['results/',tt,char(extension)];
estimates = geolocateall(char(manifest),char(extension),0,1)
dlmwrite(char(resultfile),estimates,' ','newline','pc');
quit;
```

15.8 CHANGES IN CBG_SOL.M

```
%% run constraint-based geolocation using speed of internet constraints
%% expects one file per target
%% target.list should be a file listing all the target files (see
%% geolocate.m for format)
%% writes the results to estimates soi (see geolocateall.m for format)
%% wraps geolocateall, which creates its own output files with .soi
%% extension
```

```
function[]=cbg_soi(target)
tt = target;
extension = '.txt';
manifest = ['inputs/',tt, '.txt'];
resultfile = ['results/',tt,char(extension)];
estimates = geolocateall(char(manifest),char(extension),0,-1)
dlmwrite(char(resultfile),estimates,' ','newline','pc');
%estimates = geolocateall(target_info,char(extension),0,-1)
%dlmwrite(char(resultfile),estimates, ' ');
quit;
```

15.9 CHANGES IN AUTOMATETEST.JAVA

```
System.out.println("<br>"+ "-----Constraint Based Geolocation-----" + "<br>");
```

```
String cbg_lat = "";
String cbg_lon = "";
```

```
//String CBG_HOME = System.getenv("CBG_HOME");
```

```
try{
    // Open the file that is the first
    // command line parameter
    FileInputStream fstream = new FileInputStream("/home/bilal/tulip/results/"+ip+".txt");
    // Get the object of DataInputStream
    DataInputStream in = new DataInputStream(fstream);
    BufferedReader br = new BufferedReader(new InputStreamReader(in));
    String strLine;
    //Read File Line By Line
    while ((strLine = br.readLine()) != null) {
```

```

// Print the content on the console
// System.out.println("\n\n\n\n"+strLine+"\n");

Scanner scanner = new Scanner(strLine);
scanner.useDelimiter(" ");
if ( scanner.hasNext() ){

    String id = scanner.next();
    cbg_lat = scanner.next();
    cbg_lon = scanner.next();
}
else {
    System.out.println("Empty or invalid line. Unable to process.");
}
}
//Close the input stream
in.close();
}catch (Exception e1){//Catch exception if any
    System.err.println("Error: " + e1.getMessage());
}

System.out.println("<br><br><br>"+ "Latitude through CBG is : "+cbg_lat+"<br>");
System.out.println("Longitude through CBG is : "+cbg_lon+"<br><br><br><br>");
#*****Forming Table*****
System.out.println("<table border=1>");
System.out.println("<caption> Comparison of Lat/Lons </caption>");
System.out.println("<tr>");
System.out.println("<th> Lat/Lon </th>");
System.out.println("<th> Apollonius </th>");
System.out.println("<th> Trilateration </th>");
System.out.println("<th> CBG </th>");
System.out.println("<tr>");
System.out.println("<tr>");
System.out.println("<th> Latitude </th>");
System.out.println("<td> "+Apollonius_circle.y+" </td>");
System.out.println("<td> "+LLPos.y+"</td>");
System.out.println("<td> "+cbg_lat+" </td>");
System.out.println("<tr>");
System.out.println("<tr>");
System.out.println("<th> Longitude </th>");
System.out.println("<td> "+Apollonius_circle.x+" </td>");
System.out.println("<td> "+LLPos.x+"</td>");
System.out.println("<td> "+cbg_lon+"</td>");
System.out.println("<tr>");
System.out.println("</table>");
System.out.println("<br><br><br>");
#*****Writing CBG Result in XML File*****

dos.writeBytes("<item>\n");
dos.writeBytes("<link></link>\n");
dos.writeBytes("<title> CBG </title>\n");
dos.writeBytes("<lat>"+cbg_lat+"</lat>\n");
dos.writeBytes("<lon>"+cbg_lon+"</lon>\n");
dos.writeBytes("<region> North America</region>\n");
dos.writeBytes("<type>Target</type>\n");
dos.writeBytes("<subject>CBG</subject>\n");
dos.writeBytes("<rtt></rtt>\n");
dos.writeBytes("</item>\n");

```

15.10 CHANGES IN GetPingDataPL.java

```
    FileWriter outFile = new FileWriter("/home/bilal/tulip/inputs/"+site+".txt");
    PrintWriter out = new PrintWriter(outFile);
    out.println("0 0 1 0");
    String rttm = null;
    String latt = null;
    String lonn = null;
    rttm = parser.minv;
    latt = frame.monPL[i].getLat();
    lonn = frame.monPL[i].getLon();
    // Write text to file
    if(rttm == null || latt == null || lonn == null)
    {
    }
    else if(rttm.length()==0||latt.length()==0||lonn.length()==0)
    {
    }
    else{
        out.println(latt+" "+lonn+" "+rttm+" 1");
    }
    out.close();

#*****Calling Client.pl*****
Runtime.getRuntime().exec("perl //home//bilal//tulip//client.pl "+site);
```

15.11 SERVER_TULIP.PL

```
use IO::Socket;
my $server = IO::Socket::INET->new(
    Listen => 50,
    LocalAddr => 'localhost',
    LocalPort => 5053,
    Proto => 'tcp'
) or die "Can't create server socket: $!";
my $count=0;
while(true) {
    my $client = $server->accept;
    while (<$client>) {
        if($count==0)
        {$target=$_;
        $target=substr( $target , 0 , -1 );
        $count++;
        }
        elsif($count==1)
        {open FILE, ">results/$target.txt" or die "Can't open: $!";
        print FILE $_;
        $count++;}
        else
        { print FILE $_;
        }
    }
}close FILE;
$count=0;
```

15.12 SERVER_CBG.PL

```
use IO::Socket;
my $server = IO::Socket::INET->new(
    Listen => 50,
    LocalAddr => 'localhost',
    LocalPort => 5053,
```

```

Proto => 'tcp'
) or die "Can't create server socket: $!";
my $count=0;
while(true) {
my $client = $server->accept;
while (<$client>) {
if($count==0)
{
$target=$_;
$target=substr( $target , 0 , -1);
$count++;
}
elsif($count==1)
{
open FILE, ">results/$target.txt" or die "Can't open: $!";
print FILE $_;
$count++;
}
else
{ print FILE $_;
}
}
close FILE;
$count=0;
}

```

15.13 CLIENT_TULIP.PL

```

use IO::Socket;
my $target=@ARGV[0];
my $server = IO::Socket::INET->new(
PeerAddr => 'localhost',
PeerPort => 5054,
Proto => 'tcp'
) or die "Can't create client socket: $!";
print $server "$target\n";
open FILE, "/home/bilal/tulip/inputs/$target.txt";
while (<FILE>) {
print $server $_;
}
close FILE;

```