BGP Path Advertising of an Indian ISP

In partial fulfillment of the course
BITS C335 Computer Projects

Indu R
2010A7PS010G

Table of Contents

Acknowledgement

I would like to express my sincere gratitude to my mentor for this project Mr. T.S.R.K. Prasad for his guidance, which helped me immensely in completing this project. His constant support encouraged me to successfully complete the project and draft this report.

1. Introduction

In this project, we use a looking glass to obtain data regarding the flow of traffic in the Internet at the level of ISPs (or Autonomous Systems). We then create a Java program – **ISPView**, to visualize this data as graphs, to enable a user to analyze this data with ease.

The flow of the report is as follows: Chapter 2 deals with all the theoretical background necessary to understand this project. The next chapter, chapter 3 talks about the Java program itself, giving details about the packages and classes used and the back-end database. Chapter 4 highlights the functioning of ISPView by exploring its User Interface and the various functionalites. The final chapter presents the conclusions derived and the scope of future work that is possible.

2. Theoretical Background

2.1 Internet Exchange Points (IXPs)

The Internet connects several billions of people spread across millions of private, public, academic, business, and government networks. Internet exchange points (IXPs) are a very crucial part of the infrastructure, which supports the functioning of the Internet. An IXP is a physical infrastructure through which Internet service providers (ISPs) exchange Internet traffic between their networks (autonomous systems).

IXPs reduce the portion of an ISP's traffic, which must be delivered via their upstream transit providers, thereby reducing the average per-bit delivery cost of their service. Furthermore, the increased number of paths learned through the IXP improves routing efficiency and fault-tolerance.

2.2 Border Gateway Protocol (BGP)

Border Gateway Protocol (BGP) is a standardized exterior gateway protocol designed to exchange routing and reachability information between autonomous systems (AS). Most Internet service providers must use BGP to establish routing between one another. Therefore, even though most Internet users do not use it directly, BGP is one of the most important protocols of the Internet.

A unique AS Number (ASN) is allocated to each Autonomous System for use in BGP routing. The Border Gateway Protocol does not involve traditional metrics, but makes routing decisions based on path, network policies and/or rule-sets configured by a network administrator. Thus, the Border Gateway Protocol plays a key role in the overall operation of the Internet and is involved in making core routing decisions.

2.3 National Internet Exchange of India (NIXI)

The National Internet Exchange of India (NIXI) is a non-profit company established in 2003 to provide neutral Internet Exchange Point services in India. Its main purpose is to facilitate exchange of domestic Internet traffic between the peering ISP members. This enables more efficient use of international bandwidth, saving foreign exchange. It also improves the Quality of Services for the customers of member ISPs, by avoiding multiple international hops and thus reducing latency.

2.4 Looking Glass

BGP Looking Glass servers are computers on the Internet running one of a variety of publicly available Looking Glass software implementations. A Looking Glass server (or LG server) is accessed remotely for the purpose of viewing routing info. Essentially, the server acts as a limited, read-only portal to routers of whatever organization is running the Looking Glass server.

2.5 The data source

Like most IXPs today, NIXI also gives public access to its looking glass data. It is this data, which we use for the program. To understand how this information is obtained and stored, let us take an example. We use the command

show ip bgp neighbors 218.100.48.88 advertised-routes

to list out the routing and connection data of the traffic sent out from the ISP with the address 218.100.48.88. This command produces results which consists of several lines of data which typically look like

```
*>   1.22.206.0/24     218.100.48.71                0 55410 45528 45528 45528 45528 i
```

Here, the first entry is the network within the ISP, which sent out the packet. The second entry is the IP address corresponding to the ISP, which is the next hop on the path of the packet. The third entry is the weight associated with the path. The numbers following this are the AS Numbers (ASNs) of the ISPs, which fall on the path of the packet.

3. Implementation

3.1 Packages Used

3.1.1 Apache HTTPClient

Although the java.net package provides basic functionality for accessing resources via HTTP, it doesn't provide the full flexibility or functionality needed by many applications. HTTPClient seeks to fill this void by providing an efficient, up-to-date, and feature-rich package implementing the client side of the most recent HTTP standards and recommendations. We use this library to connect to the NIXI Looking Glass server and get the data.

3.1.2 jsoup

jsoup is a Java library for working with real-world HTML. It provides a very convenient API for extracting and manipulating data, using the best of DOM, CSS, and jquery-like methods. jsoup implements the WHATWG HTML5 specification, and parses HTML to the same DOM as modern browsers do. It can be used to scrape and parse HTML from a URL, file, or string, find and extract data using DOM traversal. This library is used in the project to parse the response received from the NIXI Looking Glass server.

3.1.3 H2

H2 is a relational database management system written in Java. It can be embedded in Java applications or run in the client-server mode. A subset of the SQL (Structured Query Language) standard is supported. The main programming APIs are SQL and JDBC. It is possible to create both in-memory tables, as well as disk-based tables. Tables can be persistent or temporary. Table level locking and multi-version concurrency control are implemented. This is the DBMS used to implement the backend database used by the program.

3.1.4 GraphStream

GraphStream is a graph handling Java library that focuses on the dynamics aspects of graphs. Its main focus is on the modeling of dynamic interaction networks of various sizes. GraphStream also allows storing any kind of data attribute on the graph elements: numbers, strings, or any object. Moreover, in addition, GraphStream provides a way to handle the graph evolution in time. This means handling the way nodes and edges are added and removed, and the way data attributes may appear, disappear and evolve. This is the library, which is used to represent the data as a graph.

3.2 The Database

The java program relies on the database - ISPView, created on H2 to provide it the required data based on the user's inputs. This database consists of 4 tables, the details of which have been given below.

3.2.1 ISPNames

This table contains information about the ASNs and ISP Names. This information is used in displaying the graph in a user friendly way.

a. **ASNUM** - The AS Number of the ISP
b. **SHORTNAME** - The short name or acronym for the ISP
c. **LONGNAME** - The expansion of the ISP's short name, if any

3.2.2 IPTOASNUMDB

This table maps IP addresses of the ISPs to their ASNs.

a. **IP** - The IP address of the ISP
b. **ASNUM** - The ASNumber of the ISP

3.2.3 VERSIONDB

The java program is capable of periodically updating the database by adding a newer batch of data from the NIXI Looking Glass servers. These batches of data are tracked by enumerating them with a version number. This table keeps track of the versions of the data present in the database.

a. **TIME** - The timestamp of the version
b. **VERSION** - The version number

3.2.4 PATHDB

This is the main table, which stores the path information.

a. **VERSION** - The version number of the path
b. **ID** - A unique number identifying the path
c. **SOURCE** - The source of the data obtained
d. **REGION** - The location of the NIXI IXP being studied
e. **ASNAME** - The name of the ISP being studied
f. **ISSENT** - A Boolean value indicating if the path is for data sent (true) or received (false)
g. **NETWORK** - The IP address of the network which sent out the packet
h. **HOPS** - The total number of hops on the path
i. **HOPx** - ASN of the xth hop on the path

3.3 Software Classes

3.3.1 MainFrame

This class extends the JFrame class and creates the GUI (graphical user interface). This frame is divided into 3 panels. The first contains a JTabbedPane with 2 panels

1. Run a query - contains the inputs to create the graph
2. Tools - contains buttons to access the various tools like updating and pruning

The second contains the viewer for the graph and the third contains the functionalities used to further explore the graph.

3.3.2 GraphClass

This class contains the function CreateTree, which creates the graph based on the user's inputs. It also contains the helper functions, which access the database for the data required. Additionally there are functions used to Zoom in or out of the graph and take a snapshot of the graph.

3.3.3. ToolsClass

This class contains the functions, which implement two of the most important tools - updating the database and pruning it.

3.3.4. UpdateThread

This class implements the class Runnable to create a thread which will run the process of updating the database as a background process that runs parallel to the main program.

3.3.5. PruneThread

This class also implements the class Runnable to create a parallel thread for the pruning process, similar to that for the updating process.

3.3.6. ResizeThread

This class implements the class Runnable to ensure that the viewer for the graph created is rendered as required at all times. This thread runs throughout the duration of the program.

3.3.7. NoResultException

This is a custom exception which is thrown when the database doesnt have information for the requested set of inputs.

3.3.8. NotifyUser

This class is used to display messages to the user.

3.4 UML Diagrams

3.4.1 Class Diagram

A class diagram is a type of static structure diagram that describes the structure of a system by showing the system's classes, their attributes, operations (or methods), and the relationships among the classes. Figure 3.1 represents all the eight classes used in the software with their relationship with other classes. The classes also show their major methods used.

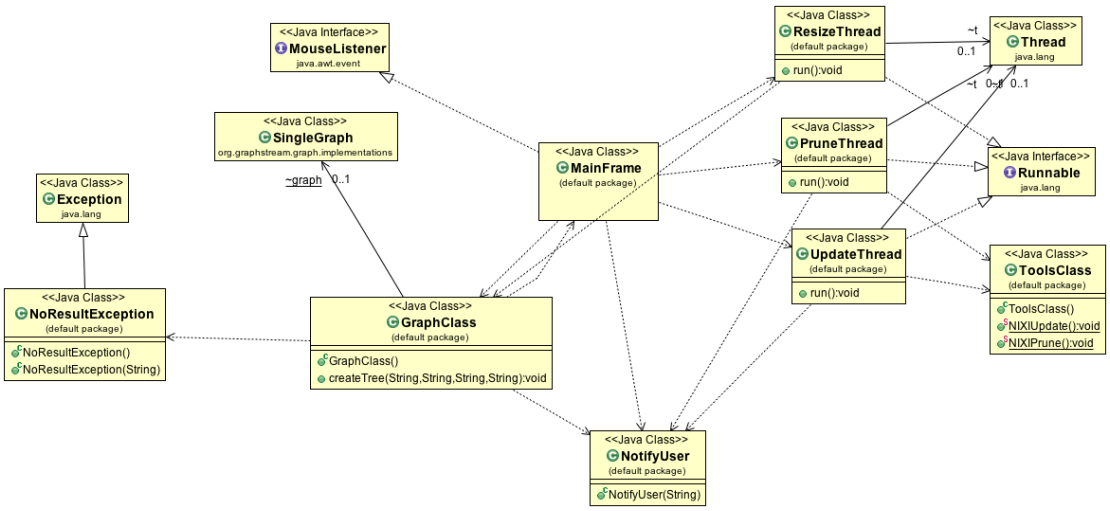


Fig 3.1 Class Diagram for ISPView

3.4.2 Sequence Diagram

The UML Sequence diagram is used to model implementation details of a scenario of the system. The Sequence diagram represents an interaction, which is a set of messages exchanged among objects within a collaboration to effect a desired operation or result.

3.4.2.1 Sequence Diagram for creating a graph

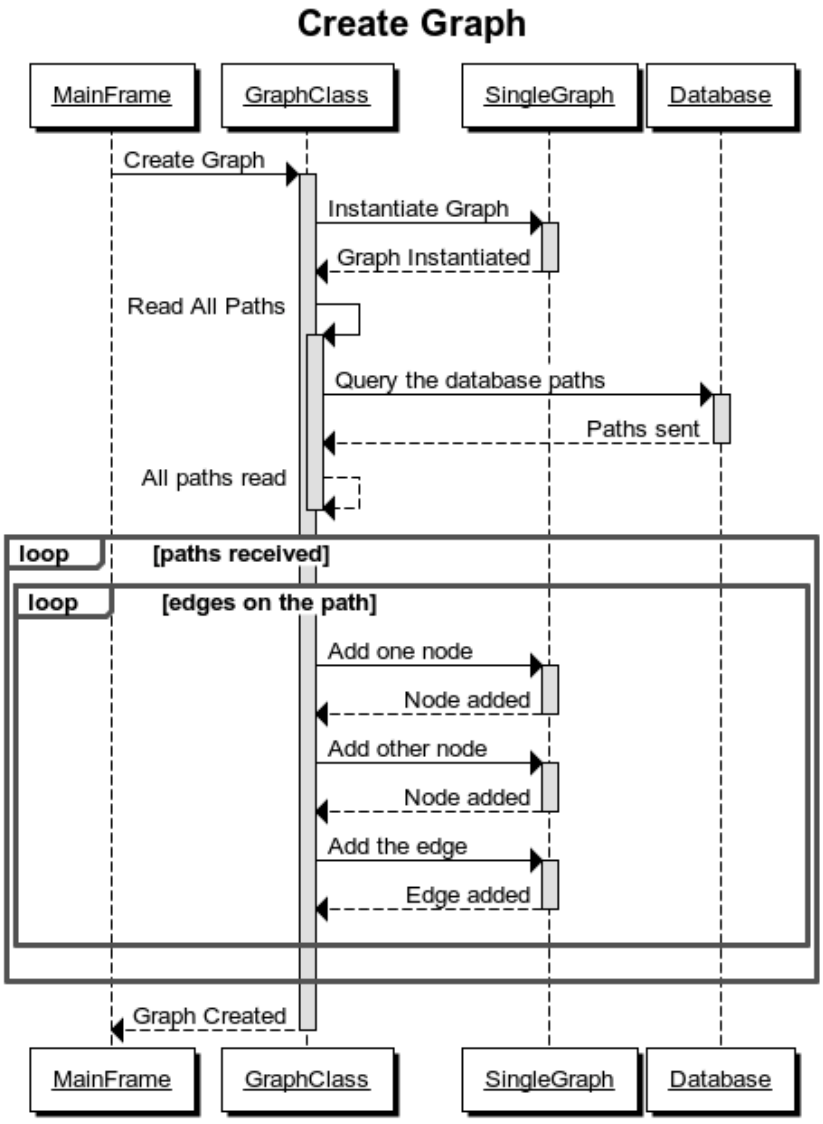


Fig 3.2 Sequence Diagram for creating a graph

3.4.2.2 Sequence Diagram for updating the database

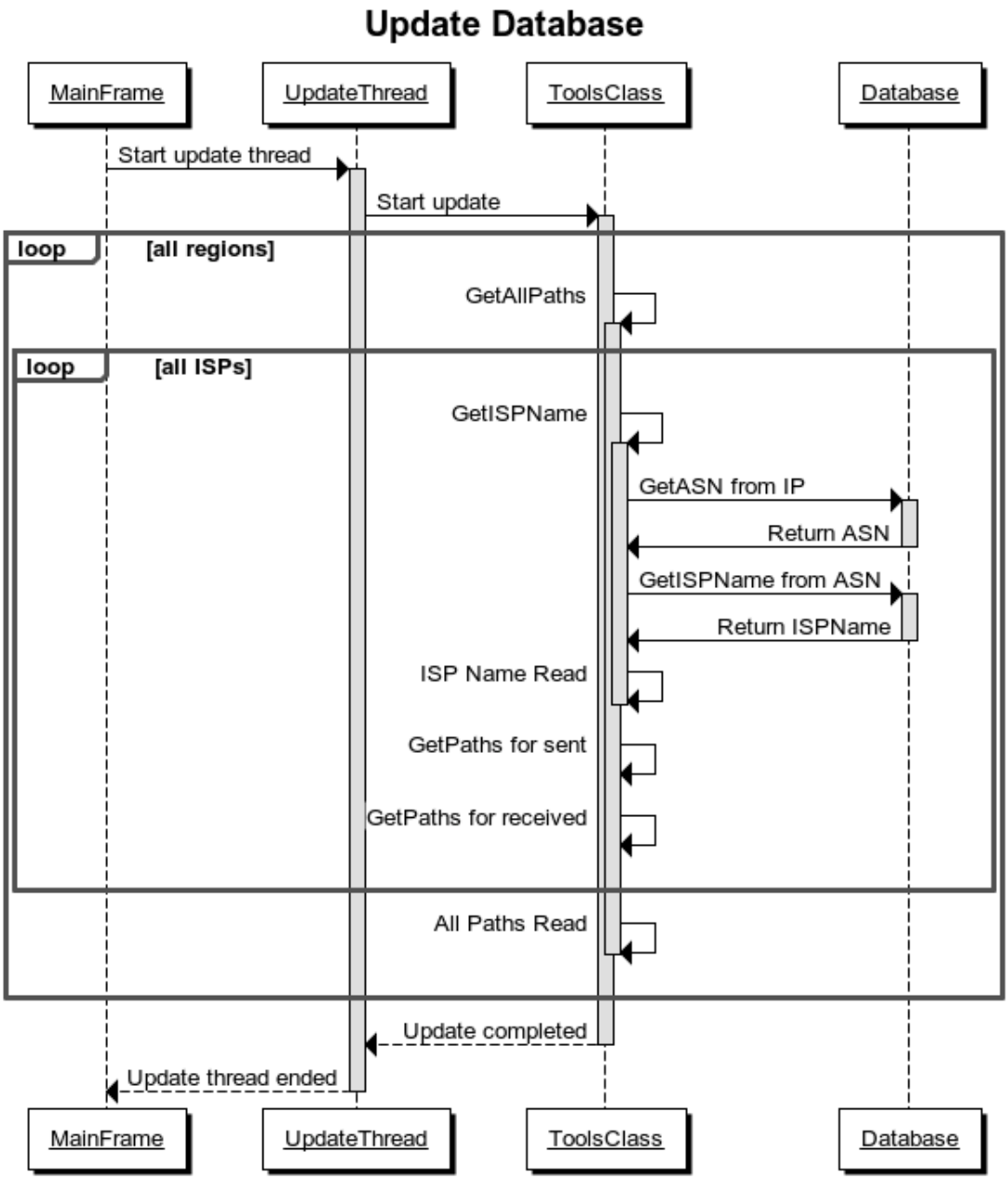
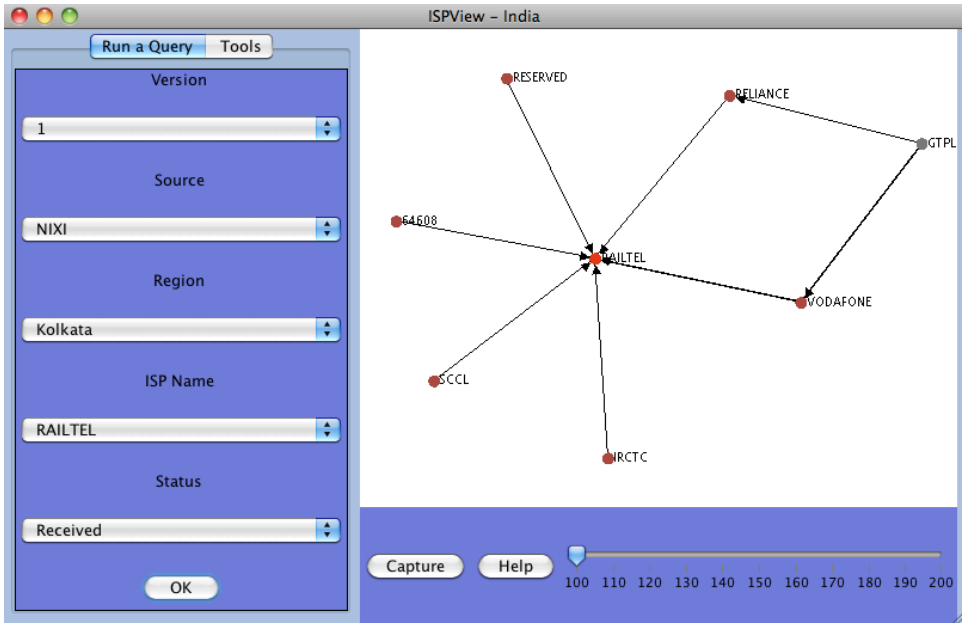


Fig 3.2 Sequence Diagram for updating the database

4 Using ISPView

4.1 Understanding the User Interface



The panel on the left has two tabs:

- **Run a Query** – to create the graph
- **Tools** – support functions for the application

The region on the right is where the graph is displayed. It also contains the additional functionalities associated with the graph.

4.2 Creating the graph

You can create a graph by choosing your input for the following parameters from the 'Run a query' tab on the left:

1. **Version** – The database maintains several batches of the information. These batches are enumerated with a 'version'. Higher the version, more recent is the data.
2. **Source** – It refers to the source of the information.
3. **Region** – It is the region for which the data has been picked up.
4. **ISP** – It is the Name of the ISP, which you want to study.
5. **Status** – It decides the type of traffic studied – sent from/ received at the chosen ISP.

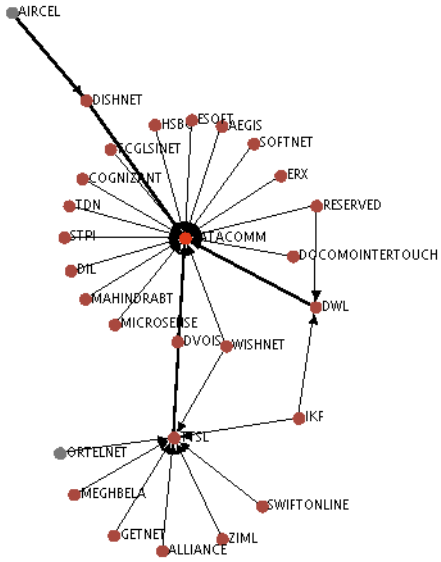
After choosing all these inputs, click 'OK' to generate the graph. You will be able to see the generated graph on the right side.

4.3 Understanding the graph

- **Nodes** – The nodes represent ISPs and are labeled by the name of the ISP they represent. They are coloured according the their distance from the ISP being studied. The colour varies from a bright red for the chosen ISP and moves towards duller shades as we move away.

- **Edges** – Each edge represents traffic flowing between two ISPs. The direction of the arrows indicates the flow of traffic. The thickness of the edges is indicative of the amount of traffic on the particular edge – the thickness increases proportionately with the frequency of usage.

Let us take the example of the following graph:



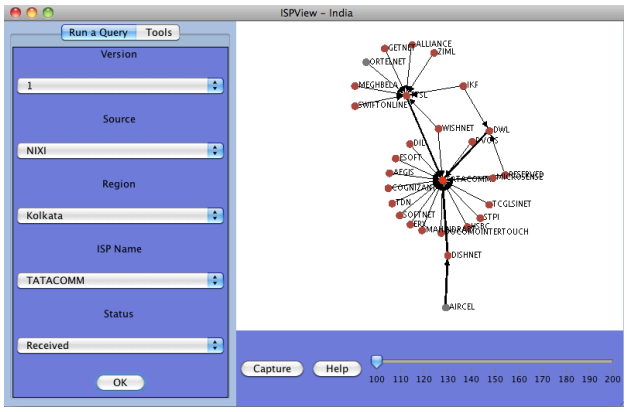
This graph shows information for the Tatacomm ISP since the node labeled Tatacomm is the brightest. The directions of the arrows show us that the traffic studied here is the one received at Tatacomm. From the colours of the nodes we can understand Airtel is further away from Tatacomm than Dishnet. And finally, from the thickness of the paths we can understand that the path from DWL to Tatacomm is used much more frequently than most of the other paths.

4.4 Taking a screenshot

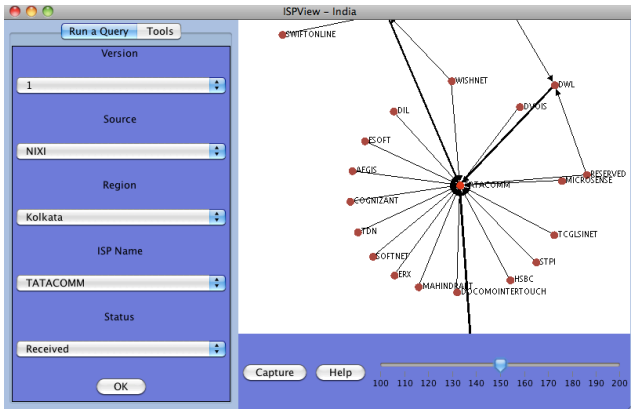
If you want to take a snapshot of the graph generated to study later, you can use the '**Capture**' button below the graph. Clicking this button saves a snapshot of the graph as a .png file in the same directory as the application.

4.5 Navigating through the graph

Zooming - Once you have created the graph, you might want to focus on certain parts of the graph. To do this, you can use the slider on the bottom right part of the UI.

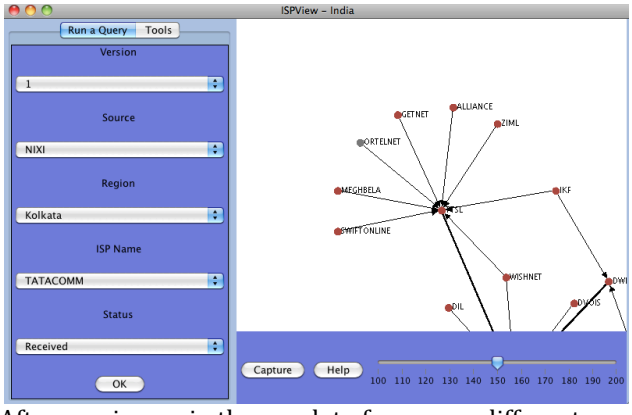


Before Zooming in



At 150% Zoom in

Panning – To move around the graph, click once somewhere inside the graph. Then use the arrow keys to move around within the graph.



After moving up in the graph to focus on a different area

4.6 Updating the database

This process will add a new batch of information to the database used by the application. To do this you need to make sure you are connected to the Internet.

To begin the process, click the **'Update Database'** button on the 'Tools' tab in the panel on the left. The update will take about 30 to 60 minutes to complete, depending on your connection. During this time, you can continue using the application to generate and study graphs.

4.7 Pruning the database

This operation keeps only latest version of the information, deleting all the older versions. This is useful when the size of the database becomes very large.

To begin the process, click the **'Prune Database'** button on the 'Tools' tab in the panel on the left. The process takes very little time and can be done anytime other than in the middle of an update.

4.8 Viewing documentation

The **'View Documentation'** button provides a link to the documentation of the source code. On clicking this button, the documentation opens up in the default browser.

5. Conclusion

We started this project by understanding the structure and functioning of an IXP. Studying the NIXI Looking Glass Server and the data it had to offer followed this. The next stage was designing the backend database and choosing an appropriate way to implement it. After this each of the functionalities was developed and tested iteratively with the final step being integration of all the features into a simple and intuitive user interface. The result of this process is the application - ISPView, which enables a user to understand routing at the ISP level by visualizing the data as a dynamic graph which he/she can explore using the various features of the application.

The way ahead for the development of ISPView would be to incorporate data from several different IXPs including those outside India. Another addition could be the incorporating the analysis of the IP addresses, which are the source of the traffic. This would help us understand the patterns of traffic flow on the Internet better.

References

NIXI India Looking Glass. www.nixi.in/lookingglass.php

CISCO. IXP Design. ftp://ftp-eng.cisco.com/pfs/isp.../2-ixp-design.pdf

Norton, William B. "Internet service providers and peering." Proceedings of NANOG. Vol. 19. 2001.

Govindan, Ramesh, and Anoop Reddy. "An analysis of Internet inter-domain topology and route stability." INFOCOM'97. Sixteenth Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings IEEE. Vol. 2. IEEE, 1997.

Chang, Hyunseok, et al. Towards capturing representative AS-level Internet topologies. Vol. 30. No. 1. ACM, 2002.

Appendix 1: Java codes to build the database

The following snippets of code can be used to build the tables ISPTOASNUM and ISPNames. The other remaining tables are maintained by the code for ISPView.

ISPNames:

```
public CreateASNumDB()
{
      try
      {
        Class.forName("org.h2.Driver");
        Connection conn = DriverManager.getConnection("jdbc:h2:~/COP", "sa", "");
        Statement stat = conn.createStatement();

        stat.execute("create table ISPNames(ASNum varchar(255) primary key, ShortName
varchar(255), LongName varchar(255))");

        BufferedReader fstream = new BufferedReader(new FileReader("Asnum.txt"));
         String s;
         s=fstream.readLine();
         while(s!=null)
         {
                String x[] =s.split(" ");
                StringBuffer Result[] = new StringBuffer[3];
                for(int i=0; i<x.length; i++)
                {
                        if(i==0)
                        {
                                Result[0]=new StringBuffer(x[0].trim());
                        }
                        else if(i==1)
                        {
                                while(x[i].length()==0)
                                        i++;
                                Result[1]=new StringBuffer(x[i].trim());
                        }
                        else
                        {
                                if(Result[2]==null)
                                {
                                        Result[2]=new StringBuffer();
                                        Result[2].append(x[i].trim());
                                }
                                else
                                        Result[2].append(" " +x[i].trim());
                        }

                }

                stat.execute("insert into ISPNames values('"+Result[0]+"',
'"+Result[1]+"', '"+Result[2]+"')");

                s=fstream.readLine();
        }
        fstream.close();
      }
      catch(Exception e)
      {
        e.printStackTrace();
      }
}
```

ISPTOASNUMDB

```
	static void SetUpDB() throws Exception
	{
	  Class.forName("org.h2.Driver");
	   Connection conn = DriverManager.getConnection("jdbc:h2:~/COP", "sa", "");
	   Statement stat = conn.createStatement();

	   try
	   {
	          stat.execute("drop table IpToAsnumDB");
	   }
	   catch(Exception e)
	   {
	          System.out.println("");
	   }
	   stat.execute("create table IpToAsnumDB(IP varchar(255) primary key, ASNum
varchar(255))");

	   conn.close();

	}

	static void GetAllPaths() throws Exception
	{
	   SetUpDB();
	   Class.forName("org.h2.Driver");
	   Connection conn = DriverManager.getConnection("jdbc:h2:~/COP", "sa", "");
	   Statement stat = conn.createStatement();

	   HttpClient client = new HttpClient();
	   String url = "http://203.190.131.164/lg/";

	     // Create a method instance.
	   PostMethod method = new PostMethod(url);

	   method.setRequestBody("query=summary&protocol=IPv4&addr=&router=NIXI+Mumbai");


	   try
	   {
	       // Execute the method.
	          int statusCode = client.executeMethod(method);


	          //Parse the response
	          InputStream Body = method.getResponseBodyAsStream();
	          Document doc = Jsoup.parse(Body, null, url);
	          Element codeElement = doc.getElementsByTag("CODE").first();
	          //Elements links = codeElement.getElementsByTag("B");
	          Elements links = codeElement.getElementsByTag("A");

	          StringBuffer QueryString = new StringBuffer("insert into IpToAsnumDB
values('");
	          int lookForASNum = 0;
	          for (Element link : links)
	          {
	                 if(links.indexOf(link)<5)
	                        continue;
	                 try
	                 {
	                        int x = Integer.parseInt(link.text());
	                        if(lookForASNum==1)
	                               lookForASNum++;
	                        else if(lookForASNum==2)
	                        {
	                               QueryString.append(link.text()+"')");
	                               System.out.println(QueryString);
	                               stat.execute(QueryString.toString());
	                               QueryString = new StringBuffer("insert into
IpToAsnumDB values('");
	                               lookForASNum=0;
	                        }
```

```
                        }
                        catch(NumberFormatException e)
                        {
                                QueryString.append(link.text()+"','");
                                lookForASNum=1;
                        }

                }
            }
            catch (HttpException e)
            {
                System.err.println("Fatal protocol violation: " + e.getMessage());
                e.printStackTrace();
            }
            catch (IOException e)
            {
                System.err.println("Fatal transport error: " + e.getMessage());
                e.printStackTrace();
             }
            finally
            {
                // Release the connection.
                method.releaseConnection();
            }
}
```