Visualizing VDOT Traffic Pattern Data

Master's Project Final Report
Project Presentation Date: Nov 14, 2008

Author: Rajat K Singh
Email: rsingh@cs.odu.edu

Project Advisor: Dr. Michele Weigle
Email: mweigle@cs.odu.edu

Department of Computer Science
Old Dominion University

TABLE OF CONTENTS

Acknowledgment

I feel privileged in expressing my deep sense of gratitude and sincere appreciation to Dr. **Michele Weigle**, Assistant Professor, Department of Computer Science, Old Dominion University, for devoting her precious time and constant support to the project. I express my sincere thanks for her keen interest, valuable guidance and constant encouragement during the whole process of developing this project. Her vast experience and in-depth knowledge of Vehicular Network has been most critical in making this project successful.

I like to convey my thanks to Mr. Greg Goebel IT Systems Engineering, VDOT, for his constant support while my project was in research phase. His valuable input and unperturbed answers to my question, made this project, an interesting learning experience.

My sincere thanks to Computer Science department of Old Dominion University for the opportunity and resources to accomplish this project and make it a success.

Abstract

The study of Vehicular Ad-Hoc Networks (VANETs) is a recent area of research interest in the networking field. Scholars and researchers are still establishing new patterns and protocols to enable communication between vehicles on the road. Most experiments are based on set of simulations (*e.g.* using ns-2, OPNET, Qualnet, etc.) or test beds. This project is based around a tool to compare the test bed results with real traffic data.

This project retrieves data from the Archived Data Management System (ADMS) Virginia. This is a system established by the Virginia Department of Transportation (VDOT) to provide access to archived data collected by traffic monitoring on inter-state roads in Virginia. It provides data about traffic volumes, speeds and allows for planning evacuations and maintenance. My project involves obtaining traffic data from ADMS Virginia and displaying the results using Google maps.

There are three phases to this project:

- Data Gathering – This phase downloads the traffic data from the previous day in the form of comma separated files every morning at 2 A.M.

- Data Processing – In this phase of my project, downloaded data is imported into a database that is used for generating reports for different parameters such as speed vs. time or occupancy vs. time.

- Data Visualization – This phase uses the Google Maps API to show the exact position of the various monitoring stations along I-64 Eastbound using the latitude and longitude data provided by ADMS and allows users to view graphs of various statistics at those locations.

This approach will allow researchers to study traffic data and observe how patterns change throughout 24 hours and between days. This will benefit researchers in developing more realistic traffic patterns for simulations.

1. Introduction

VDOT collects real-time traffic, weather and incident data from various Virginia interstates and roads and provides it to researchers who have valid accounts with them. The aim of this project is to utilize the real time traffic data provided by VDOT and present it into condensed graphical format which makes the data comparable and meaningful for analysis. This tool is built to transform raw data into visualized format thereby making it more readable for researchers and observers. This project utilizes only traffic data and not incident or weather data.

2. Motivation

Networking between vehicles means that information can be exchanged between vehicles without human intervention. This is ongoing research, and the main purpose is to help prevent traffic accidents and losses. Various universities, government agencies and research-based firms are continuously conducting researches and experiments to understand and implement the VANET concepts. These organizations and bodies are experimenting with various factors to determine the cons and pros of their research applications. At present, there are a number of simulation tools available to these researchers so that they can create various test beds in order to evaluate their theories. Also, many government and private institutes have established hardware set ups around intersections or local roads of cities to facilitate this research with current data feeds. And these organizations are providing live data feeds (e.g. VDOT) to various researchers in order to conduct their planned experiments. However the simulators or other testing tools require normalized data to form a particular scenario and present the test results.

Just like many other states, Virginia has a government agency which collects live traffic data everyday. This data could be extremely useful for comparisons

between test beds and the live traffic data patterns, in order to close the distance between experiment and reality. Moreover, traffic data collected from these government agencies is a lot of raw numerical data. It would be helpful to researchers to have tools to organize and present the data in graphical form. This project is a tool which will utilize huge amount of real time I-64 traffic data and present it into meaningful information.

3. Background

This section describes the background for my project. The Virginia Department of Transportation (VDOT) is a government organization responsible for the state level construction and maintenance of roads and highways. Traffic monitoring is a specialized branch of VDOT that installs software and hardware along side of the highways to track traffic patterns and utilize that for future planning. The data collected through traffic monitoring is real time data and is very useful for other road-related and traffic-related research. The Archived Data Management System (ADMS) is a web-based system that provides the traffic monitoring data collected by VDOT to other organizations, universities and researchers.

3.1 Virginia Department of Transportation (VDOT)

The Commonwealth of Virginia has established an organization which is responsible for building, maintaining and operating the state's roads, bridges and tunnels[1], called the Virginia Department of Transportation (VDOT). This organization also provides funds to various airports, seaports and rail transportation through Commonwealth Transportation Board (CTB). The Central Office of VDOT is located in Richmond and is headquarters for approximately 30 operational and administrative units. The Commonwealth Transportation Board acts as board of directors for the organization and guides the department's work.

[1] www.virginiadot.org

The board members are constituted of 17 members. These members are appointed by the governor and approved by the General Assembly.

VDOT aggressively participates in research projects as a continuous effort for future improvement of the roads. It openly provides the data and results obtained from its own research and projects to various universities and organizations and individuals to utilize in their own projects. One such project is called ADMS.

3.2 Traffic Monitoring

Traffic Monitoring is one of the programs that VDOT handles. In this program, traffic data is collected from sensors of inductive dual loop detectors in or along roads, interstates, bridges and other sources. These are also called stations in VDOT terminology. The collected data is used to calculate the average number of vehicles that traveled each segment of road.

3.2.1 Inductive Dual Loop Detector

A station consists of a complete mechanism for collecting data, including hardware and software and networking to the main data center. The main component of a station is an inductive dual loop detector.

An inductive dual-loop detector is formed by two consecutive single-loop detectors installed a few meters away [2]. Inductive dual loop is created using a conductive wire which is configured in a way that it is placed on a plane surface as two parallel paths lying adjacent to each other.

A single loop detector can record the volume and occupancy of traffic at a given instance of time. However a dual-loop detector is capable of recording the time used for a vehicle to traverse from the first loop to the second loop. Since the distance between the two loops is predetermined, a dual-loop detector can

calculate traffic speed fairly accurately. By applying the calculated speed from the dual loops and the single loop measured lane occupancies, the traffic speed and volume is collected.

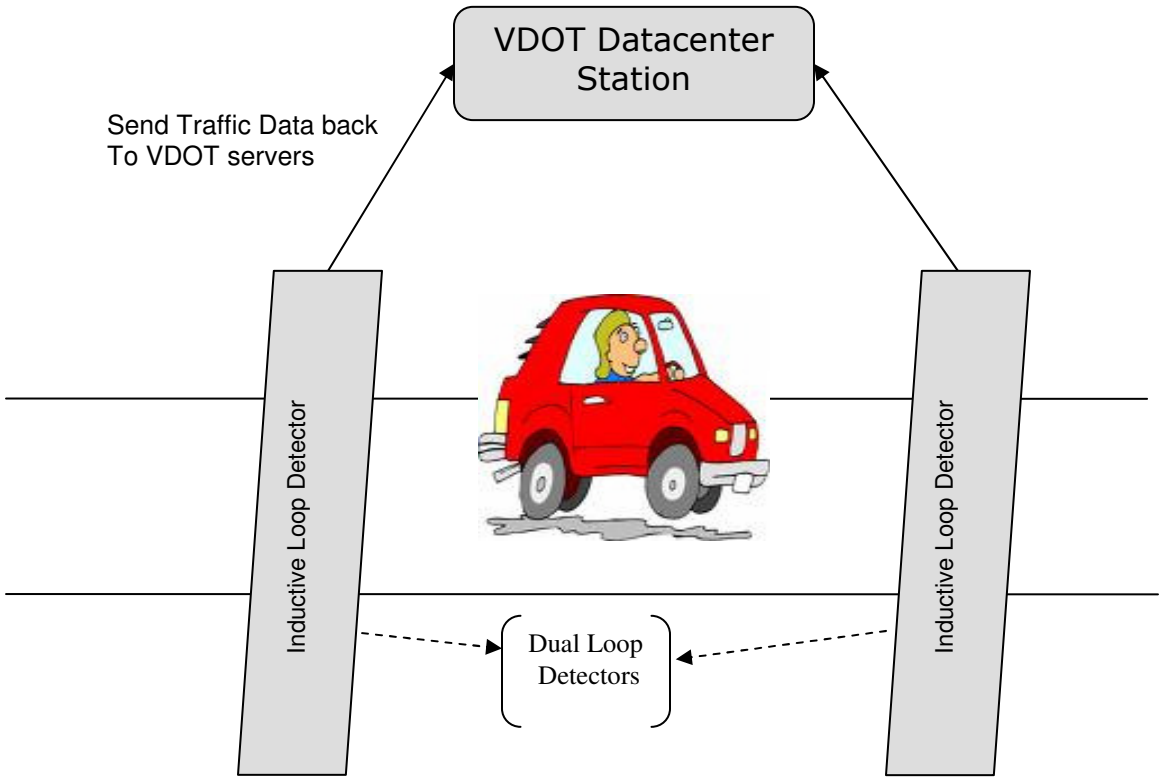

Figure 1 Dual Loop Detectors Mechanism

Figure 1 shows the working of a VDOT station. The dual loop detectors are configured as a set of two single loop detectors. These inductive loops collect data (e.g. volume of traffic, capacity, speed etc.) and send back it continuously to the VDOT datacenter. Every midnight, VDOT uploads this data to ADMS servers.

3.3 Archived Data Management System (ADMS)

The Archived Data Management System (ADMS) enables the use of system-enhanced archived traffic operations-related data for transportation applications such as planning and mobility performance measurement, improved operational effectiveness, and decision support. ADMS is the place where much of the research of the Smart Travel Laboratory (STL) is implemented as well as the place that people can go to access the data resources of the STL. The Federal Highway Administration (FHWA) and VDOT originally funded ADMS Virginia as an operational test. VDOT continues to sponsor this project and is leading the effort with the team members of the Virginia Transportation Research Council (VTRC) and the University of Virginia (UVA) Center for Transportation Studies (CTS). UVA has subcontracted the software development part of the project to Open Roads Consulting, Inc. (ORCI) [4].

ADMS Virginia provides query capability to users of the system. Authorized users may log onto the system through a standard web interface and query the database for historic traffic, weather, and incident information for specific routes, segments, or detector stations at user specified dates, times, and levels of aggregation.

This is web-based and Geographical Information System-based (GIS-based) software. ADMS provides their collected data to students, professors, scholars who are involved directly or indirectly to projects related to roads and transportation.

Data is collected everyday on Virginia interstates and gets uploaded every midnight to VDOT severs. The data amount is extremely huge and is available in raw format. The raw data is complex and can be difficult to understand.

VDOT-installed stations are equipped with networking facilities. These stations continuously send collected data back to the data servers at the ADMS data warehouse. ADMS shuts down its web-server access to the public at night during

a maintenance window and uploads the traffic data of the day onto their servers for public use. At a given time when ADMS is open to the public, it contains data as old as one day.

ADMS provides raw data in Excel sheet format or XML format. It provides following types of data:

- Traffic Data
- Incident Data
- Weather Data

4. Methodology

Using the valid account credentials, traffic data is retrieved from ADMS. However some part of the data could be incorrect. Data can be bad for various reasons. The most common reason is that stations along the roadsides become inoperable or malfunction due to wear and tear, road maintenance, road accidents, depreciated hardware, etc. For example, there are 56 stations on Interstate 64 and only 8 stations gather sensible data. Other stations have either dysfunctional hardware or produce garbage data due to malfunctioning.

The project is divided into 3 stages and each stage is atomic in its nature. These stages are:

- Data Gathering
- Data Processing
- Data Visualization

4.1 Phase 1 - Data Gathering:

The first phase of the project is to gather data into the local hard disk. There is a large amount of data available on VDOT servers thus making the process of downloading quite complex. This project is designed to download data every night from the ADMS data center shortly after midnight. ADMS updates their database every midnight and uploads the data received from the stations to the database server. The system is designed to download traffic data for previous day and keeps a backup for last 30 days of real-time data.

ADMS does not support XML-based web services or FTP servers at this point, but provides data only through their online web-based system. This means one has to manually navigate through the required web page to access data. And this data is available on a secure channel. Therefore, this project used the mechanism of Screen Scraping. Screen scraping is a process where specialized code is written to parse the output of the web browser on the screen. The code that scrapes the screen acts like a human working through the web page to get the required result from the webpage and save it or re-use it as input into another program.

For this project, I wrote a script which is scheduled every night, 2 hours after midnight, to scrape the website of ADMS using secured and approved credentials and store its copy on the web server at ODU-CS. Data is saved in form of comma separated value (CSV) files.

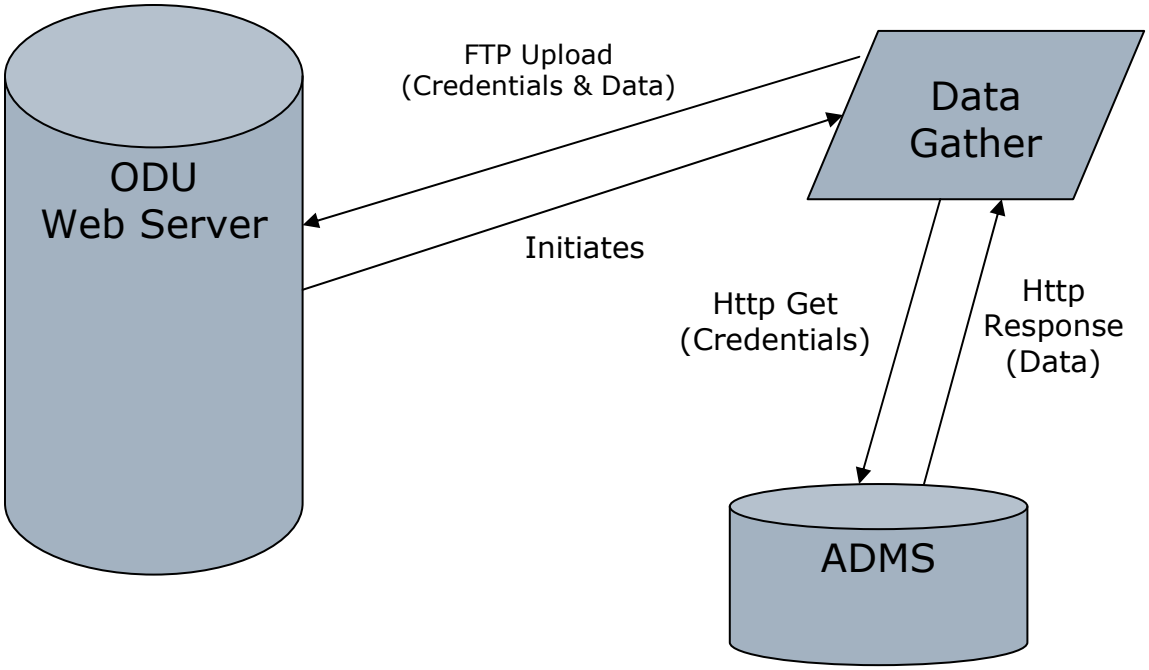

Figure 2 Data Gathering Phase

The data gathering phase is explained through Figure 2. Every night at 2:00 am the ODU web server starts a job that initiates the screen scraping script. The script initiates the screen scraping and uses HTTP to transfer the credentials and get a response from the ADMS data center. When script receives the comma separated (CSV) data files from the ADMS server, it stores the file onto the ODU web server. The script that runs on the ODU server is stored locally but does not have access to save data files directly onto the server. However, for this project FTP access is provided to one of the ODU FTP servers. The data gathering script keeps the downloaded data into cache blocks and uploads that data to the ODU FTP server. The access to ODU server is also credential based and is not open anonymously.

4.2 Phase 2 – Data Processing

Figure 3 shows a snapshot of CSV file that is the result of phase 1. In phase 2 this file is used by the project.

	A	B	C	D	E	F	G	H	I	J	K	
1	Station ID	Day	Date	Time	Interstate	Direction	Mile Marker	Lane Type	Volume (vehicle counts)	Occupancy (%)	Speed (mph)	Sa
2993	51	Fri	11/7/2008	9:15:00	64	EB	282.2	Norm	262	6.8	43.3	
2994	51	Fri	11/7/2008	9:20:00	64	EB	282.2	Norm	265	7.2	42.3	
2995	51	Fri	11/7/2008	9:25:00	64	EB	282.2	Norm	244	5.6	48.1	
2996	51	Fri	11/7/2008	9:30:00	64	EB	282.2	Norm	266	6.6	47.2	
2997	51	Fri	11/7/2008	9:35:00	64	EB	282.2	Norm	280	7.6	41.9	
2998	51	Fri	11/7/2008	9:40:00	64	EB	282.2	Norm	274	7.2	44.1	
2999	51	Fri	11/7/2008	9:45:00	64	EB	282.2	Norm	255	6.2	45.9	
3000	51	Fri	11/7/2008	9:50:00	64	EB	282.2	Norm	239	7.4	44	
3001	51	Fri	11/7/2008	9:55:00	64	EB	282.2	Norm	228	6.6	43.3	
3002	51	Fri	11/7/2008	10:00:00	64	EB	282.2	Norm	224	5.8	44.1	
3003	51	Fri	11/7/2008	10:05:00	64	EB	282.2	Norm	243	6.2	44.9	
3004	51	Fri	11/7/2008	10:10:00	64	EB	282.2	Norm	230	6.2	42.7	

Figure 3 Snapshot of Comma Separated File

The second phase of the project is a web-based application which has access to the last 30 days backup of the database on the ODU FTP server. This application is always available and is hosted on the ODU web server. It is a research tool, accessible to everyone and thus there are no credentials restrictions on this application.

The client interface is designed using AJAX and HTML. The aim while designing the web interface was to keep it simple, easy to use and provide as much information to the user as possible. As discussed above, the data gathering phase downloads a large amount of raw data from the ADMS data server which contains garbage values from dead stations as well as data related to weather and incidents. VDOT stations are designed to collect data and send it to server. At the end of first phase we have raw data which needs processing and data filtering.

Phase 2 is responsible for data processing. Depending on user input, data is mined and a kept in cache as long as data is completely filtered according to the

user. At present, meaningful data is obtained through two parameters only. These are:

- Date
- Station

Date could be any date from last 30 days, because the system maintains the back up for last 30 days only. If more resources are given, then the range for date selection could be increased accordingly.

The station is the part of interstate that user wants to get data and graphs for. This project is focused on a section of Interstate 64, which has 56 stations installed by VDOT, however at the given time period only 8 stations are providing correct data. Thus at present, the options for a user are limited to 8 stations out of which only one can be selected.

After applying the above filters, the amount of data remains still huge and complex in order to extract meaningful information out of it. It contains weather and incident data, which this project does not require. The system is designed to generate traffic information out of the remaining chunk of data. The following are three reports that the system generates:

- Time vs. Speed
- Time vs. Occupancy
- Time vs. Volume

The reports are generated as scatter graphs. Time data is divided into 5 minute intervals for the date chosen by the user. Speed data is in format of miles per hour. Occupancy is given in terms of percentage for the station for which the data is collected per 5 minutes of time interval. Similarly, volume is the number of vehicle counts on the given station average between 5 minutes of interval. The system displays a report as per the filters provided by the user.

4.3 Phase 3 – Data Visualization

Data in itself could be confusing if it is not represented in a format which is clear and concise. The third phase in the project is to process data and present the extracted information with graphics. As explained earlier, on the interstate there are 8 stations only at the moment that are active and gather useful information. The exact location of these stations in terms of longitude and latitude are available from the VDOT data web server. This information is available on the ADMS website in the format of XML. This is a good example of wide range of code diversity in the project. The second phase had a backend in form of CSV files. In the third phase, data is stored in XML format. Using the given latitude and longitude and Google Maps API, the project provides a visualized view of the interstate and points out the stations along the interstate. This sort of visualization is extremely important in providing as many factors as possible for test bed settings. For example, the traffic pattern near a tunnel would be different than the traffic pattern near exits.

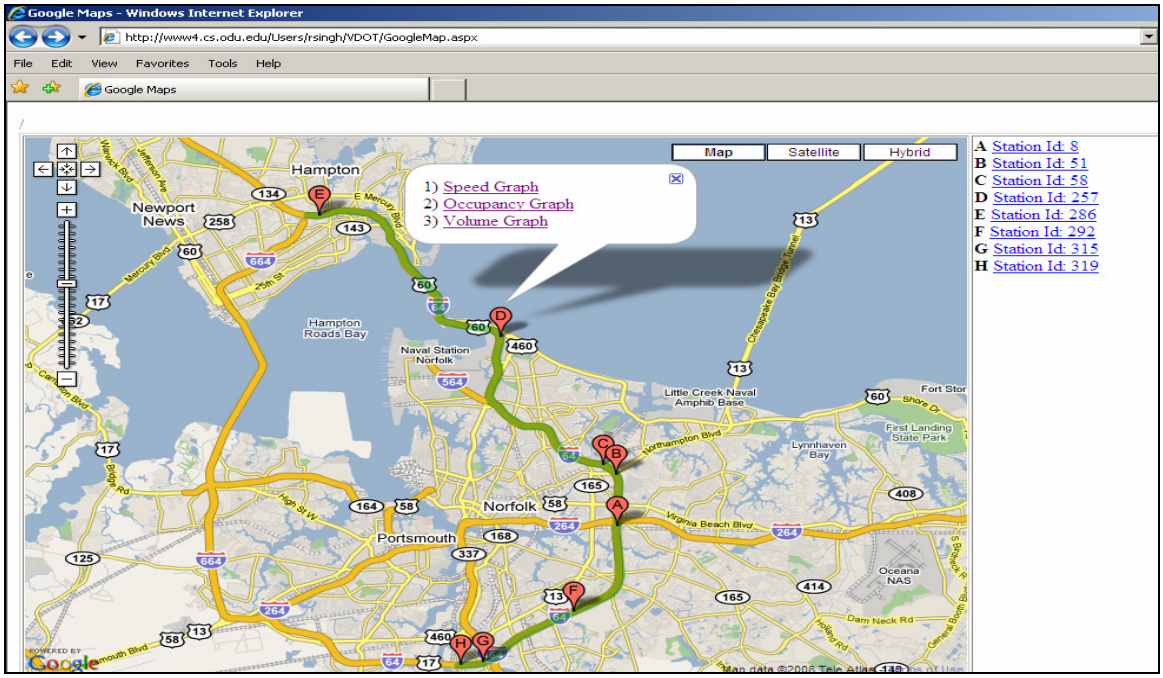

Figure 4 Stations shown along I-64

In order to show the exact location on Google Map, the system parses the XML provided by ADMS and points out the stations onto the map. This is to visualize the stations on the webpage for user to view. The screen shot is given in Figure4.

The following are some screen shots showing the working of the project.

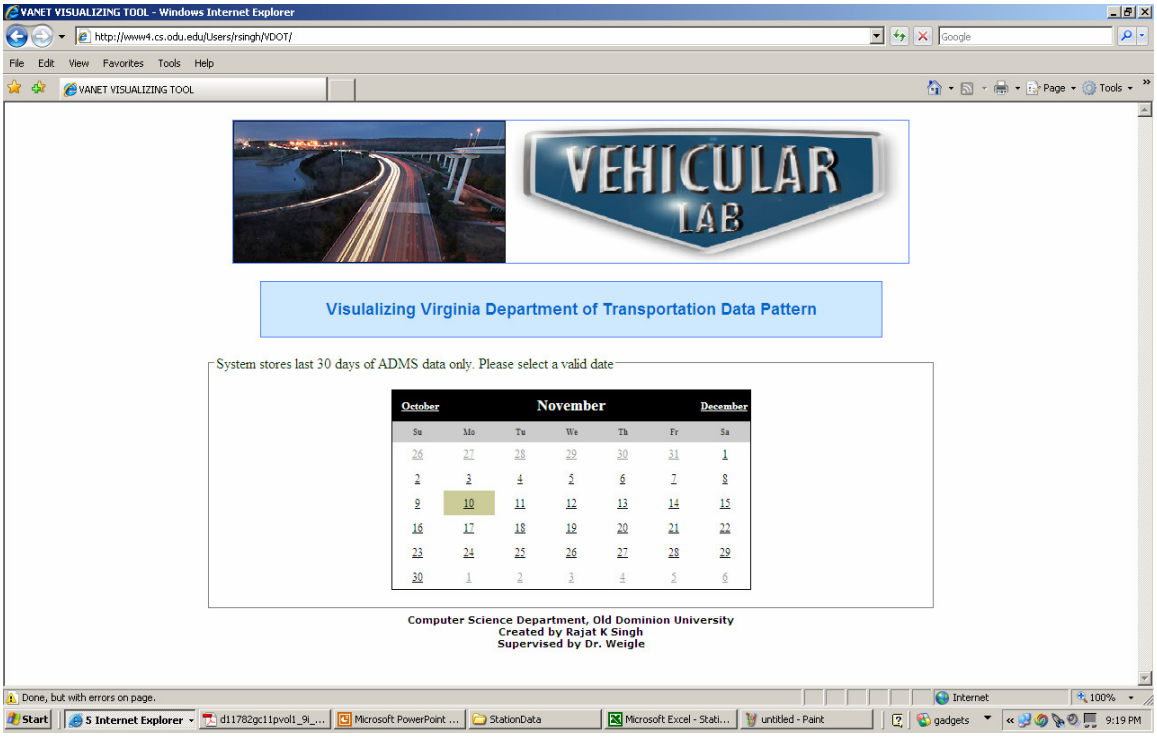

Figure 5 Home Page

Figure 5 is the home page of the tool. It provides an option for the user to select the date for generating graphs. The calendar is disabled for all other days except the last 30 days.

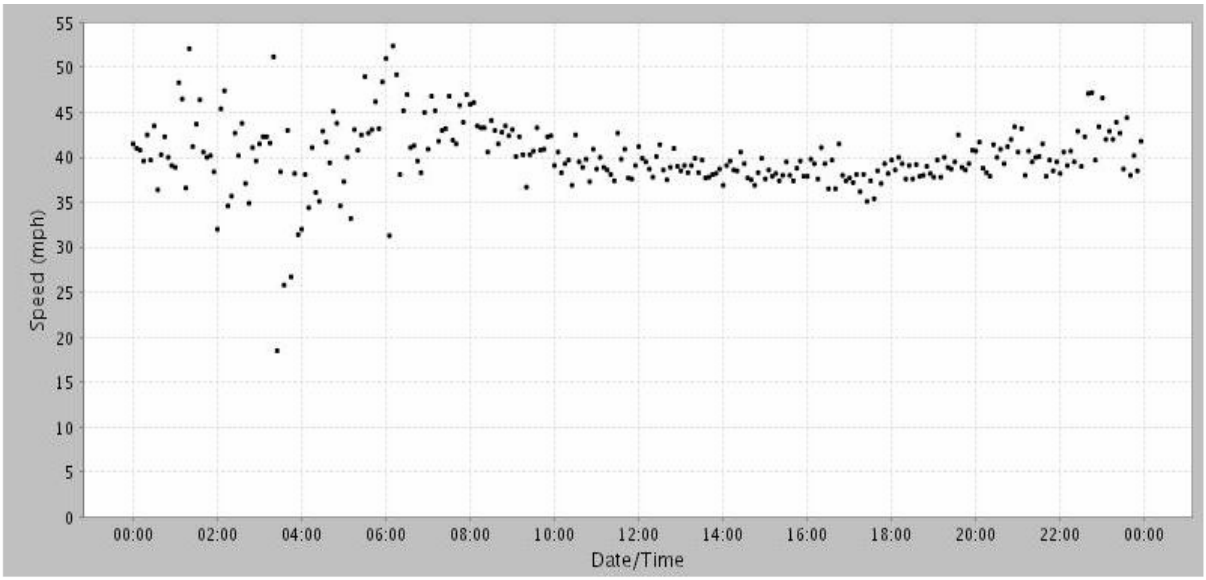

Figure 6 Time vs. Speed Graph

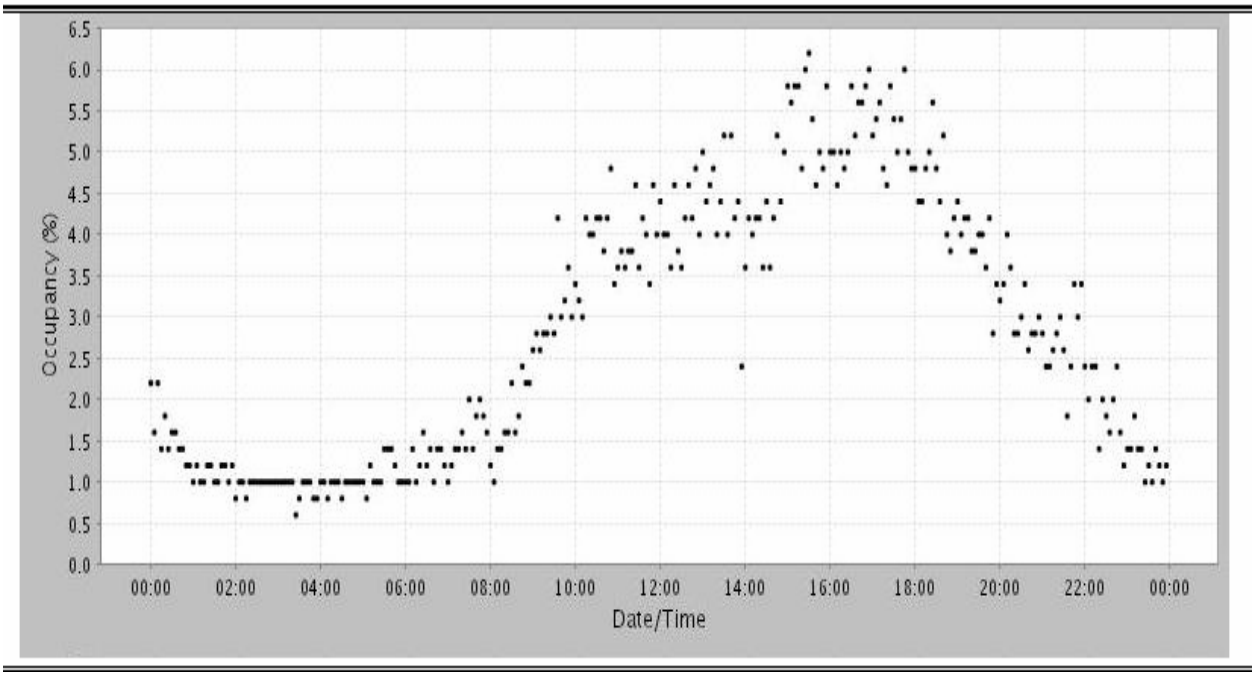

Figure 7 Time vs. Occupancy Graph

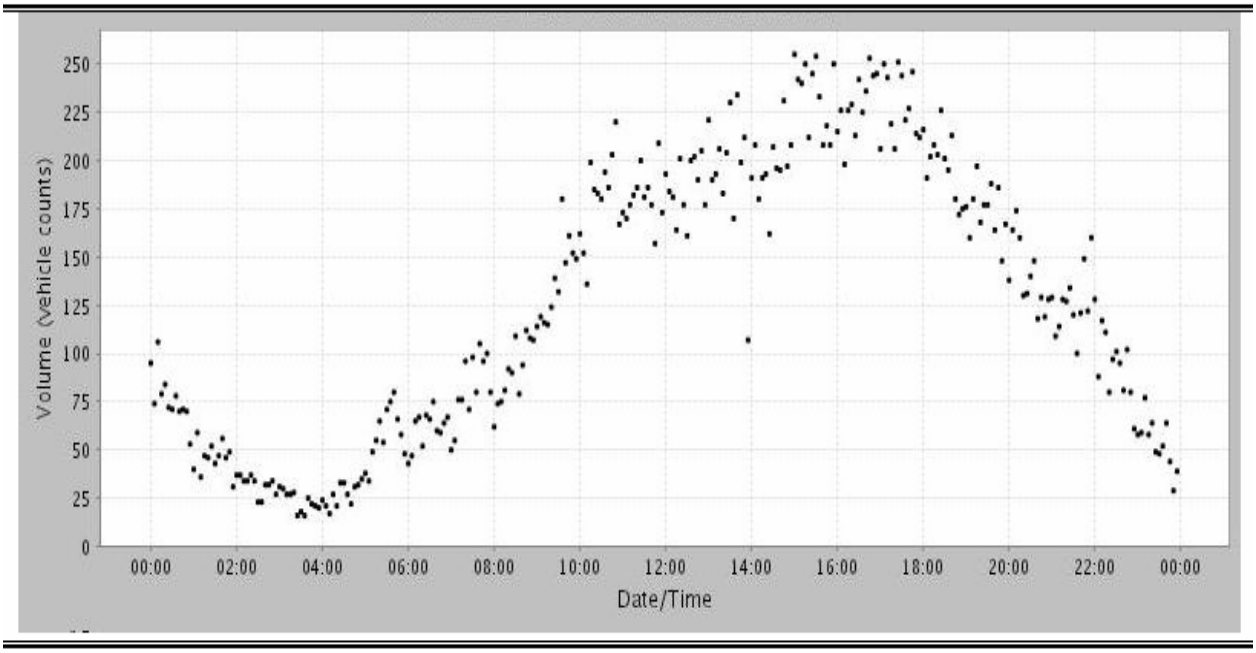

Figure 8 Time vs. Volume

Figure 6, Figure 7 and Figure 8 show the graphical presentation of the data, *i.e.* Time vs. Speed, Time vs. Occupancy and Time vs. Volume, respectively.

5. Limitations

VDOT traffic data patterns can be visualized and used for future research through this project. This project is potentially an important tool for VANET research. However, the project is built to consider data based on 5 minute intervals as timeline. This could be more relaxed to 1 minute or 1 hour intervals. In order to implement this, we have two options, either save data files from ADMS with 1 min and/or 1 hour timeline or save data files with 1 minute time interval and then while processing the data file calculate according to the time interval required. However, in both these options, the constraint of space and complexity is huge. With 5 minutes of time interval the size of one file is more than 800 KB and there are 30 such files that system must save. With 1 minute of time interval the size is much bigger and thus would take up huge space on the

FTP server. To upload and download huge files from FTP servers take up a lot of bandwidth and makes the whole process more time consuming. More over with such huge amount of data in cache, making calculations becomes complex in terms of system memory and processing it.

6. Summary

Visualizing VDOT Traffic Pattern Data is a web-based solution and a tool for vehicular network researchers. It aims at providing meaningful information from live traffic data feeds by presenting it into visualized format. The graphical meaningful data can be used by the researchers and scholars to compare and contrast their laboratory analysis and generate reports on the basis of their findings.

7. Future Work

This project is has a lot of potential for enhancements. The project is implemented for a specific roadway *i.e.,* I-64 Eastbound in Hampton Roads. There is a scope that the selection of filters could be more relaxed and provide more options to select from. The project could be expanded in terms of including more sections of roads and more intersections and more number of days rather than only last 30 days of data.

Another area for enhancing the project is to utilize the available weather and incident data into account and present them into meaningful reports. This kind of data also plays a vital importance in VANET research.

References

[1] Gradinescu, V. Gorgorin, C. Diaconescu, R. Cristea, V. Iftode, L.. "Adaptive Traffic Lights Using Car-to-Car Communication". Vehicular Technology Conference. April 2007.

[2] Yinhai Wang, Nancy L. Nihan. "Can Single-Loop Detectors Do the Work of Dual-Loop Detectors?" Journal of Transportation Engineering, Vol. 129. March/April 2003

[3] Haas, H. and Brown, A. "Web Services Glossary", World Wide Web Consortium, HTML, 2002.

[4] Ramkumar Venkatanarayana, Smart Travel Laboratory. "ADMS Virginia". TMC Applications of Archived Data Operational Test. December 2003

Appendix A – Website Setup

With permission of Dr. Weigle, the Systems Group of the Computer Science Department provided me with access on the department's web server called Cyclops. The Systems Group granted me an FTP permission to use the server. I uploaded my website files on the cyclops.cs.odu.edu server via FTP under my account. Then I created a virtual directory of my project folder, by following steps:

1. Open **cyclops.cs.odu.edu** on an explorer.
2. Click the link "**Click here to enable your Web Application**" from the page.
3. It will ask for credentials. Use credentials of Computer Science network.
4. Select the project you want to open for public from drop-down box. My project folder is named **VDOT** which contains all the required files.
5. Finally hit the button "**Make this directory a web application**"

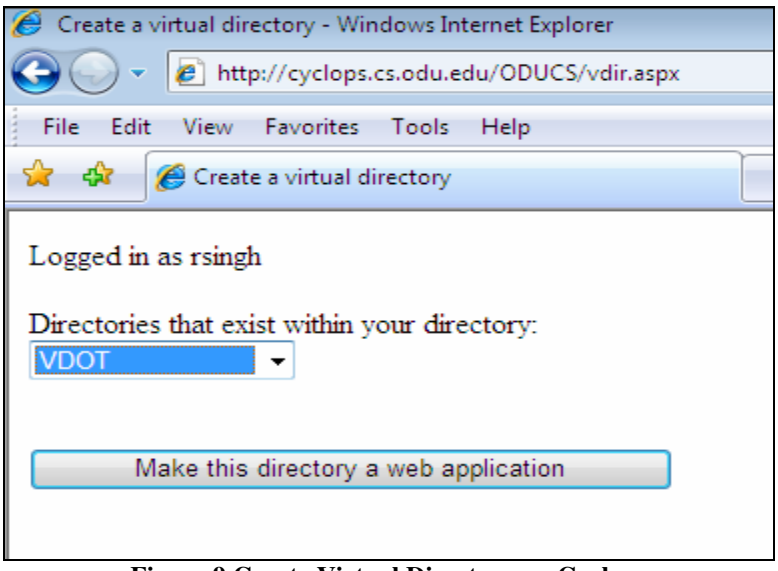

Figure 9 Create Virtual Directory on Cyclops

This will make the selected project folder a virtual directory on the department's window server. The server takes some time to configure and within a day the website is hosted and live for public to use.

The following code is the code behind the options that a user selects in order to get a visualized graph for the real time data. User can select the date, the station and the type of graph. Instead of typing, the user can easily make selections through a visualized calendar, Google map of stations with exact physical locations, and option to select the type of graph

```
  protected void Calendar1_SelectionChanged(object sender, EventArgs e)
    {
        if( (Calendar1.SelectedDate >= System.DateTime.Today) ||
(Calendar1.SelectedDate < System.DateTime.Today.AddDays(-30)))
        {
            panelError.Visible = true;
            panelDate.Enabled = false;
            return;
        }
        Session["Date"] = Calendar1.SelectedDate.Date.ToString();
        Response.Redirect("GoogleMap.aspx");
        PutDateInBox();
    }


    <script type="text/javascript">
    //<![CDATA[

        if (GBrowserIsCompatible()) {


            var iwform = 'Enter details:<br>'
                 + '><form action="#" onsubmit="process(this); return
false" action="#">'
                 + '  <textarea name="data" rows="5"
cols="40"><\/textarea><br>'
                 + '  <input type="submit" value="Submit" />'
                 + '<\/form>';



            // this variable will collect the html which will
eventualkly be placed in the side_bar
            var side_bar_html = "";

            // arrays to hold copies of the markers and html used by
the side_bar
            var gmarkers = [];
            var htmls = [];
            var i = 0;

    // Display the map, with some controls and set the initial location
              var map = new GMap2(document.getElementById("map"));
              map.addControl(new GLargeMapControl());
```

```
                map.addControl(new GMapTypeControl());
                map.setCenter(new GLatLng(36.910921, -76.266403), 11);


            // A function to create the marker and set up the event window
                function createMarker(point, htmls, labels, name) {
                    // use a custom icon with letter A - Z
                    var letter = String.fromCharCode("A".charCodeAt(0) +
(gmarkers.length));
                    var myIcon = new GIcon(G_DEFAULT_ICON,
"http://www.google.com/mapfiles/marker" + letter + ".png");
                    myIcon.printImage =
"http://maps.google.com/mapfiles/marker" + letter + "ie.gif"
                    myIcon.mozPrintImage =
"http://maps.google.com/mapfiles/marker" + letter + "ff.gif"

                    var marker = new GMarker(point, { icon: myIcon });
                    GEvent.addListener(marker, "click", function() {



         // adjust the width so that the info window is large enough for
this many tabs
                  if (htmls.length > 2) {
                  htmls[0] = '<div style="width:'+htmls.length*88+'px">' +
htmls[0] + '<\/div>';
                  }
                  var tabs = [];
                  for (var i=0; i<htmls.length; i++) {
                  tabs.push(new GInfoWindowTab(labels[i],htmls[i]));
                  }



                    marker.openInfoWindowTabsHtml(tabs);
                    });
                    // save the info we need to use later for the side_bar
                    gmarkers.push(marker)
                    // add a line to the side_bar html
                    side_bar_html += '<b>' + letter + '<\/b> <a
href="javascript:myclick(' + (gmarkers.length - 1) + ')">' + name +
'<\/a><br>';
                    return marker;
                }


                // This function picks up the click and opens the
corresponding info window
                function myclick(i) {
                    GEvent.trigger(gmarkers[i], "click");
                }


                function getValue(test, station) {
                    window.alert(test);
```

```
                window.alert(station);

            }

      // Set up eight markers with info windows

          var point = new GLatLng(36.842536, -76.19635);
          var marker = createMarker(point, ["<a
href=StationGraph.aspx?StationID=8&Graph=Speed>Graph</a>", "<a
href=StationGraph.aspx?StationID=8&Graph=Occupancy>Occupancy
Graph</a>", "<a href=StationGraph.aspx?StationID=8&Graph=Volume>Volume
Graph</a>"], ["Speed", "Occupancy", "Volume"], 'Station Id: 8')
          map.addOverlay(marker);

          var point = new GLatLng(36.873683, -76.19676);
          var marker = createMarker(point, ["<a
href=StationGraph.aspx?StationID=51&Graph=Speed>Speed Graph</a>", "<a
href=StationGraph.aspx?StationID=51&Graph=Occupancy>Occupancy
Graph</a>", "<a href=StationGraph.aspx?StationID=51&Graph=Volume>Volume
Graph</a>"],["Speed","Occupancy","Volume"], 'Station Id: 51')
          map.addOverlay(marker);

          var point = new GLatLng(36.879584, -76.20456);
          var marker = createMarker(point,["<a
href=StationGraph.aspx?StationID=58&Graph=Speed>Speed Graph</a>", "<a
href=StationGraph.aspx?StationID=58&Graph=Occupancy>Occupancy Graph</a>
click me", "<a href=StationGraph.aspx?StationID=58&Graph=Volume>Volume
Graph</a>"],["Speed","Occupancy","Volume"], 'Station Id: 58')
          map.addOverlay(marker);

          var point = new GLatLng(36.957608, -76.264172);
          var marker = createMarker(point,["<a
href=StationGraph.aspx?StationID=257&Graph=Speed>Speed Graph</a>", "<a
href=StationGraph.aspx?StationID=257&Graph=Occupancy>Occupancy
Graph</a>", "<a
href=StationGraph.aspx?StationID=257&Graph=Volume>Volume
Graph</a>"],["Speed","Occupancy","Volume"], 'Station Id: 257')
          map.addOverlay(marker);

          var point = new GLatLng(37.031671, -76.37042);
          var marker = createMarker(point,["<a
href=StationGraph.aspx?StationID=286&Graph=Speed>Speed Graph</a>", "<a
href=StationGraph.aspx?StationID=286&Graph=Occupancy>Occupancy
Graph</a>", "<a
href=StationGraph.aspx?StationID=286&Graph=Volume>Volume
Graph</a>"],["Speed","Occupancy","Volume"], 'Station Id: 286')
          map.addOverlay(marker);

          var point = new GLatLng(36.790275, -76.22194);
          var marker = createMarker(point,["<a
href=StationGraph.aspx?StationID=292&Graph=Speed>Speed Graph</a>", "<a
href=StationGraph.aspx?StationID=292&Graph=Occupancy>Occupancy
Graph</a>", "<a
href=StationGraph.aspx?StationID=292&Graph=Volume>Volume
Graph</a>"],["Speed","Occupancy","Volume"], 'Station Id: 292')
          map.addOverlay(marker);
```

```
          var point = new GLatLng(36.759103, -76.274643);
          var marker = createMarker(point,["<a
href=StationGraph.aspx?StationID=315&Graph=Speed>Speed Graph</a>", "<a
href=StationGraph.aspx?StationID=315&Graph=Occupancy>Occupancy
Graph</a>", "<a
href=StationGraph.aspx?StationID=315&Graph=Volume>Volume
Graph</a>"],["Speed","Occupancy","Volume"], 'Station Id: 315')
          map.addOverlay(marker);

          var point = new GLatLng(36.757458, -76.28753);
          var marker = createMarker(point,["<a
href=StationGraph.aspx?StationID=319&Graph=Speed>Speed Graph</a>", "<a
href=StationGraph.aspx?StationID=319&Graph=Occupancy>Occupancy
Graph</a>", "<a
href=StationGraph.aspx?StationID=319&Graph=Volume>Volume
Graph</a>"],["Speed","Occupancy","Volume"], 'Station Id: 319')
          map.addOverlay(marker);

          // put the assembled side_bar_html contents into the side_bar
div
          document.getElementById("side_bar").innerHTML =
side_bar_html;

          var request = GXmlHttp.create();
          request.open("GET", "Location.xml", true);
          request.onreadystatechange = function() {
          if (request.readyState == 4) {
          var xmlDoc = GXml.parse(request.responseText);

                  // ========= Now process the polylines ===========
          var lines =
xmlDoc.documentElement.getElementsByTagName("line");
          // read each line
          for (var a = 0; a < lines.length; a++)
              {
            // get any line attributes
            var colour = lines[a].getAttribute("colour");
            var width  = parseFloat(lines[a].getAttribute("width"));
            // read each point on that line
            var points = lines[a].getElementsByTagName("point");
            var pts = [];
            for (var i = 0; i < points.length; i++)
            {
               pts[i] = new
GLatLng(parseFloat(points[i].getAttribute("lat")),

parseFloat(points[i].getAttribute("lng")));
            }
            map.addOverlay(new GPolyline(pts,colour,width));
          }
          }
          }
         request.send(null);

    }
```

```
    // display a warning if the browser was not compatible
    else {
      alert("Sorry, the Google Maps API is not compatible with this
browser");
    }


    //]]>

      function abc()
            {
                  document.getElementById('aaa').value = "abc";
                  document.getElementById('SubmitButton').click();
            }

    </script>
     <form id="Form1" method="post" runat="server">
           
            </form>
```

The Following XML gives the station location co-ordinates in terms of longitudes and latitudes. This file is provided by ADMS.

```
<?xml version="1.0" encoding="utf-8" ?>
- <markers>
  <marker lat="36.842536" lng="-76.19635" />
  <marker lat="36.873683" lng="-76.19676" />
  <marker lat="36.879584" lng="-76.20456" />
  <marker lat="37.031671" lng="-76.37042" />
  <marker lat="36.790275" lng="-76.22194" />
  <marker lat="36.757458" lng="-76.28753" />
- <line colour="#008800" width="8" html="This is a I-64 EB Area Region">
  <point lat="37.03188433" lng="-76.37613773"
    markerIcon="/images/greenCircle.pointng" />
  <point lat="37.03164451" lng="-76.37073040" />
  <point lat="37.03243248" lng="-76.36635303" />
  <point lat="37.03435096" lng="-76.36025906" />
  <point lat="37.03489909" lng="-76.35485172" />
  <point lat="37.03448799" lng="-76.34983063" />
  <point lat="37.03298062" lng="-76.34060383" />
  <point lat="37.02958892" lng="-76.33433819" />
  <point lat="37.02869814" lng="-76.33309364" />
  <point lat="37.02397001" lng="-76.32987499" />
  <point lat="37.02150304" lng="-76.32828712" />
  <point lat="37.01807656" lng="-76.32807255" />
  <point lat="37.01643179" lng="-76.32742882" />
  <point lat="37.01341630" lng="-76.32438183" />
  <point lat="37.01207985" lng="-76.32318020" />
  <point lat="37.01050349" lng="-76.32223606" />
```

<point lat="37.00289540" lng="-76.31901741" />
<point lat="37.00200431" lng="-76.31867409" />
<point lat="37.00094184" lng="-76.31772995" />
<point lat="36.98939087" lng="-76.30691528" />
<point lat="36.98280911" lng="-76.30073547" />
<point lat="36.97019249" lng="-76.29944801" />
<point lat="36.96847811" lng="-76.29858971" />
<point lat="36.96772377" lng="-76.29747391" />
<point lat="36.96535784" lng="-76.28829002" />
<point lat="36.96347191" lng="-76.27730370" />
<point lat="36.96240890" lng="-76.26962185" />
<point lat="36.96117443" lng="-76.26696110" />
<point lat="36.96024856" lng="-76.26558781" />
<point lat="36.95880830" lng="-76.26464367" />
<point lat="36.95719655" lng="-76.26404285" />
<point lat="36.95483029" lng="-76.26417160" />
<point lat="36.94608480" lng="-76.26623154" />
<point lat="36.94474717" lng="-76.26644611" />
<point lat="36.94385540" lng="-76.26644611" />
<point lat="36.94313512" lng="-76.26627445" />
<point lat="36.93781854" lng="-76.26507282" />
<point lat="36.93565750" lng="-76.26511574" />
<point lat="36.92865942" lng="-76.26837730" />
<point lat="36.92680689" lng="-76.26902103" />
<point lat="36.92279290" lng="-76.27026558" />
<point lat="36.92148916" lng="-76.27017975" />
<point lat="36.92035696" lng="-76.26983643" />
<point lat="36.91960214" lng="-76.26957893" />
<point lat="36.91881301" lng="-76.26889229" />
<point lat="36.91768076" lng="-76.26769066" />
<point lat="36.91623970" lng="-76.26597404" />
<point lat="36.91486723" lng="-76.26451492" />
<point lat="36.91287710" lng="-76.26309872" />
<point lat="36.91109281" lng="-76.26159668" />
<point lat="36.90920553" lng="-76.25962257" />
<point lat="36.90573967" lng="-76.25455856" />
<point lat="36.90378363" lng="-76.25116825" />
<point lat="36.90347477" lng="-76.24979496" />
<point lat="36.90333750" lng="-76.23975277" />
<point lat="36.90213639" lng="-76.23666286" />
<point lat="36.90076367" lng="-76.23520374" />
<point lat="36.89846430" lng="-76.23417377" />
<point lat="36.89365943" lng="-76.23215675" />
<point lat="36.88830507" lng="-76.22880936" />
<point lat="36.88648588" lng="-76.22713566" />
<point lat="36.88542180" lng="-76.22537613" />
<point lat="36.88126833" lng="-76.21194363" />
<point lat="36.88006687" lng="-76.20503426" />
<point lat="36.87958628" lng="-76.20366096" />

```
<point lat="36.87855643" lng="-76.20151520" />
<point lat="36.87718328" lng="-76.19966984" />
<point lat="36.87584444" lng="-76.19795322" />
<point lat="36.87364732" lng="-76.19632244" />
<point lat="36.87206809" lng="-76.19550705" />
<point lat="36.87024851" lng="-76.19499207" />
<point lat="36.86829156" lng="-76.19473457" />
<point lat="36.85139785" lng="-76.19580746" />
<point lat="36.84057980" lng="-76.19640827" />
<point lat="36.82972587" lng="-76.19542122" />
<point lat="36.81529738" lng="-76.19452000" />
<point lat="36.81196470" lng="-76.19456291" />
<point lat="36.80887239" lng="-76.19490623" />
<point lat="36.80670770" lng="-76.19567871" />
<point lat="36.80320283" lng="-76.19773865" />
<point lat="36.80062562" lng="-76.20048523" />
<point lat="36.79406193" lng="-76.21310234" />
<point lat="36.78653534" lng="-76.22902393" />
<point lat="36.77804555" lng="-76.24683380" />
<point lat="36.77450500" lng="-76.25331402" />
<point lat="36.76515440" lng="-76.26511574" />
<point lat="36.76168201" lng="-76.26932144" />
<point lat="36.75996295" lng="-76.27215385" />
<point lat="36.75893150" lng="-76.27464294" />
<point lat="36.75793441" lng="-76.27777576" />
<point lat="36.75769373" lng="-76.28193855" />
<point lat="36.75748744" lng="-76.28837585"
  markerIcon="/images/redCircle.pointng" />
 </line>
 </markers>
```

Appendix B

The following is the Screen Scraping code which runs as scheduled task every night at 2:00am. This is the code which parses through web pages of ADMS. It downloads a zipped file from ADMS and uploads it via FTP onto the Cyclops server.

```
#region Screen Scrapping
        //First PAGE
        // first, request the login form to get the viewstate value

        HttpWebRequest webRequest =
WebRequest.Create("http://adms.vdot.virginia.gov/ADMSVirginiaHR/GetLogi
n.event") as HttpWebRequest;
        StreamReader responseReader = new
StreamReader(webRequest.GetResponse().GetResponseStream());
        string responseData = responseReader.ReadToEnd();
        responseReader.Close();

        // extract the viewstate value and build out POST data

        string viewState = ExtractViewState(responseData);
        //string postData =
String.Format("UserID={0}&PIN={1}&Login=Login", "00586667", "828282");
        string postData =
String.Format("__VIEWSTATE={0}&Username={1}&Password={2}&Login=Login",
viewState, "mweigle", "carN3t");


        // have a cookie container ready to receive the forms auth
cookie
        CookieContainer cookies = new CookieContainer();

        // now post to the login form
        webRequest =
WebRequest.Create("http://adms.vdot.virginia.gov/ADMSVirginiaHR/DoLogin
.event") as HttpWebRequest;
        webRequest.Method = "POST";
        webRequest.ContentType = "application/x-www-form-urlencoded";
        webRequest.CookieContainer = cookies;

        // write the form values into the request message
        StreamWriter requestWriter = new
StreamWriter(webRequest.GetRequestStream());
        requestWriter.Write(postData);
        requestWriter.Close();
```

```
        // we don't need the contents of the response, just the cookie
it issues
        webRequest.GetResponse().Close();

        // now we can send out cookie along with a request for the
protected page
        webRequest =
WebRequest.Create("http://adms.vdot.virginia.gov/ADMSVirginiaHR/DoLogin
.event") as HttpWebRequest;
        webRequest.CookieContainer = cookies;
        responseReader = new
StreamReader(webRequest.GetResponse().GetResponseStream());

        // and read the response
        responseData = responseReader.ReadToEnd();
        responseReader.Close();

        webRequest =
WebRequest.Create("http://adms.vdot.virginia.gov/ADMSVirginiaHR/GetFron
tFrame.event") as HttpWebRequest;
        webRequest.Method = "GET";
        webRequest.ContentType = "application/x-www-form-urlencoded";
        webRequest.CookieContainer = cookies;
        responseReader = new
StreamReader(webRequest.GetResponse().GetResponseStream());

        // and read the response
        responseData = responseReader.ReadToEnd();
        responseReader.Close();
        //  Response.Write(responseData);

        webRequest =
WebRequest.Create("http://adms.vdot.virginia.gov/ADMSVirginiaHR/GetStdD
ataQuery.event") as HttpWebRequest;
        webRequest.Method = "GET";
        webRequest.ContentType = "application/x-www-form-urlencoded";
        webRequest.CookieContainer = cookies;
        responseReader = new
StreamReader(webRequest.GetResponse().GetResponseStream());

        // and read the response
        responseData = responseReader.ReadToEnd();
        responseReader.Close();
        //Response.Write(responseData);

        webRequest =
WebRequest.Create("http://adms.vdot.virginia.gov/ADMSVirginiaHR/GetStat
ionDataDownload.event") as HttpWebRequest;
        webRequest.Method = "GET";
        webRequest.ContentType = "application/x-www-form-urlencoded";
        webRequest.CookieContainer = cookies;
        responseReader = new
StreamReader(webRequest.GetResponse().GetResponseStream());

        // and read the response
        responseData = responseReader.ReadToEnd();
        responseReader.Close();
```

```
        // Response.Write(responseData);
        #endregion

        webRequest =
WebRequest.Create("http://adms.vdot.virginia.gov/ADMSVirginiaHR/SaveSta
tionDataDownload.event") as HttpWebRequest;
        webRequest.Method = "POST";
        webRequest.ContentType = "application/x-www-form-urlencoded";
        webRequest.CookieContainer = cookies;

        webRequest.Headers.Add("attachment",
"filename=StationData.zip");

        // write the form values into the request message
        requestWriter = new
StreamWriter(webRequest.GetRequestStream());
        String date = null;
        if (day == "")
            date = System.DateTime.Now.Date.AddDays(-
1.0).ToShortDateString().Replace("/", "%2F");
        if (day == "Monday")
            date = System.DateTime.Now.Date.AddDays(-
2.0).ToShortDateString().Replace("/", "%2F");

requestWriter.Write("Region=HR&StartDate=06%2F12%2F1998&FeedbackLink=ma
ilto%3Astl-admins%40virginia.edu&SpaceBy=corridor&SelectCorridor=I-
64+EB&SelectDateFrom=" + date + "&SelectDateTo=" + date +
"&SelectTimeFrom=&SelectTimeTo=&TimeAggregate=5minute&SelectPNormFrom=0
.0&SelectPNormTo=1.0&UseImputedData=on&ExcludeDataQuality=on&DownloadFo
rmat=csv&SaveQuery=");

        requestWriter.Close();


        HttpWebResponse wr = null;
        int bytesProcessed = 0;
        Stream remoteStream = null;

        try
        {

            wr = webRequest.GetResponse() as HttpWebResponse;

            remoteStream = wr.GetResponseStream();

            String filename = "ftp://" + "cyclops.cs.odu.edu" + "/" +
"StationData.zip";

            FtpWebRequest ftpReq =
(FtpWebRequest)FtpWebRequest.Create(new Uri(filename));

            ftpReq.Credentials = new NetworkCredential("csnet\\rsingh",
"00571728");

            ftpReq.Method = WebRequestMethods.Ftp.UploadFile;

            Stream ftpRespStream = ftpReq.GetRequestStream();
```

```
            byte[] buffer = new byte[2024];
            int bytesRead;
            // Simple do/while loop to read from stream until
            // no bytes are returned
            do
            {
                // Read data (up to 1k) from the stream
                bytesRead = remoteStream.Read(buffer, 0,
buffer.Length);
                // Write the data to the local file
                //localStream.Write(buffer, 0, bytesRead);
                ftpRespStream.Write(buffer, 0, bytesRead);
                // Increment total bytes processed
                bytesProcessed += bytesRead;

            } while (bytesRead > 0);

            ftpRespStream.Dispose();
            remoteStream.Dispose();

        }
        catch (WebException ex)
        {

        }



          //Extract("filename", "", day);

         Extract("c:\\Inetpub\\ftproot\\rsingh\\StationData.zip",
"c:\\Inetpub\\ftproot\\rsingh\\VDOT", day);


         string fileNameDel = "c:\\Inetpub\\ftproot\\rsingh\\VDOT\\" +
System.DateTime.Now.AddDays(-31.0).ToShortDateString().Replace("/",
"_") + ".csv";
            FileInfo fileDel = new FileInfo(fileNameDel);
            if (fileDel.Exists)
                System.IO.File.Delete(fileNameDel);
            fileNameDel = "c:\\Inetpub\\ftproot\\rsingh\\VDOT\\" +
System.DateTime.Now.AddDays(-32.0).ToShortDateString().Replace("/",
"_") + ".csv";
            fileDel = new FileInfo(fileNameDel);
            if (fileDel.Exists)
                System.IO.File.Delete(fileNameDel);



    }

    private string ExtractViewState(string s)
    {
        string viewStateNameDelimiter = "Username";
        string valueDelimiter = "value=\"\"";
```

```
        int viewStateNamePosition = s.IndexOf(viewStateNameDelimiter);

        int viewStateValuePosition = s.IndexOf(valueDelimiter,
viewStateNamePosition);


        int viewStateStartPosition = viewStateValuePosition +
valueDelimiter.Length;
        int viewStateEndPosition = s.IndexOf("\"",
viewStateStartPosition);

        return
HttpUtility.UrlEncodeUnicode(s.Substring(viewStateStartPosition,
viewStateEndPosition - viewStateStartPosition));
    }


    private void Extract(string zipFileName, string destinationPath,
string day)
    {
        ZipFile zipfile = new ZipFile(zipFileName);
        System.Collections.Generic.List<ZipEntry> zipFiles =
GetZipFiles(zipfile);
        zipfile.close();
        foreach (ZipEntry zipFile in zipFiles)
        {
            if (zipFile.getName().Contains(".xml"))
                continue;
            if (!zipFile.isDirectory())
            {
                InputStream s = zipfile.getInputStream(zipFile);
                try
                {
                     Directory.CreateDirectory(destinationPath + "\\" +
                     Path.GetDirectoryName(zipFile.getName()));
                     FileOutputStream dest = new
FileOutputStream(Path.Combine(destinationPath + "\\" +
                     Path.GetDirectoryName(zipFile.getName()),
                     System.DateTime.Now.AddDays(day == "Monday" ? -2.0
: -1.0).ToShortDateString().Replace("/", "_") + ".csv"));
                    try
                    {
                        int len = 0;
                        sbyte[] buffer = new sbyte[7168];
                        while ((len = s.read(buffer)) >= 0)
                        {
                            dest.write(buffer, 0, len);
                        }
                    }
                    finally
                    {
                        dest.close();
                    }
                }
                finally
                {
                    s.close();
```

```
                }
            }
        }
    }

    private System.Collections.Generic.List<ZipEntry>
GetZipFiles(ZipFile zipfil)
    {
        System.Collections.Generic.List<ZipEntry> lstZip = new
System.Collections.Generic.List<ZipEntry>();
        Enumeration zipEnum = zipfil.entries();
        while (zipEnum.hasMoreElements())
        {
            ZipEntry zip = (ZipEntry)zipEnum.nextElement();
            lstZip.Add(zip);
        }
        return lstZip;
    }
```