Collection and Characterization of BCNET BGP Traffic

by

Sukhchandan Lally

B.Tech., Punjab Technical University, 2008

Thesis Submitted in Partial Fulfillment
of the Requirements for the Degree of
Master of Applied Science

in the
School of Engineering Science
Faculty of Applied Science

Approval

Name:	**Sukhchandan Lally**
Degree:	**Master of Applied Science**
Title of Thesis:	***Collection and Characterization of BCNET BGP Traffic***

Examining Committee:

Chair: Ash Parameswaran, Professor

Ljiljana Trajkovic
Senior Supervisor
Professor

Carlo Menon
Supervisor
Associate Professor

Lesley Shannon
Internal Examiner
Associate Professor
School of Engineering Science

Date Defended/Approved: May 25, 2012

Abstract

Measuring and monitoring traffic in deployed communication networks is necessary for effective network operations. Traffic analysis allows network operators to understand the network user’s behavior and ensure quality of service (QoS). In this thesis, we describe collection, extraction, and analysis of BGP traffic. Border Gateway Protocol (BGP) is an Inter-Autonomous System routing protocol that operates over a reliable transport protocol (TCP).

We collected real traffic from a real deployed network called BCNET. Collection of real traffic was used in the process of extraction of BGP messages and their attributes. The traffic was collected using special purpose hardware: Net Optics Director 7400, Ninjabox 5000, and the Endace DAG 5.2X card. Collected data were analyzed and compared using the Wireshark, an open-source packet analyzer. Walrus, a visualization tool, was used to visualize the graphs in three-dimensional space.

Keywords: Traffic collection; Autonomous System (AS); Border Gateway Protocol (BGP); Wireshark

To my husband and my family for all their love and support

Acknowledgements

I express my deepest gratitude to my senior supervisor, Professor Ljiljana Trajkovic for accepting me in the Communication Networks Laboratory. I would like to thank her for her constant support, guidance, insight, patience, and trust.

I would like to thank Prof. Lesley Shannon and Prof. Carlo Menon for serving on my examining committee and Prof. Ash Parameswaran for chairing the thesis defense.

My sincere thanks to my colleagues in the Communication Networks Laboratory: Rajvir Gill, Ravinder Paul, Tanjila Farah, Don Xu, Soroush Haeri, and Khaled Alutaibi for their constructive suggestions and comments. I would like to thank Nabil Al-Rousan for generating the C# code and letting me use a part of his code. I am also very grateful to BCNET and the assistance provided by Mr. Toby Wong, the technician at BCNET who made the collection of traffic traces possible.

I am blessed with such a wonderful family that stood by me with their love and prayers all through the period of my graduate studies. Big thanks to my parents Harbahadur Singh Lally and Gurpreet Lally, my siblings Harsandal Lally and Sartaj Lally, and last but the most important my husband Gurneet Purewal for his constant encouragement and support. I could not have made it without his moral, emotional, and intellectual support. Thank you all for believing in me and encouraging me to pursue greater achievements.

Table of Contents

List of Tables

List of Figures

List of Acronyms

ACK	Acknowledgement
AfriNIC	African Regional Internet Registry
AIMD	Additive-Increase Multiplicative-Decrease
APNIC	Asia Pacific Network Information Centre
ARIMA	Autoregressive Integrative Moving Average
ARIN	American Registry for Internet Numbers
AS	Autonomous System
ASCII	American Standard Code for Information Interchange
BGP	Border Gateway Protocol
CAIDA	The Cooperative Association for Internet Data Analysis
CANARIE	Canada’s Advanced Research and Innovation Network
CDF	Cumulative Distribution Function
CDPD	Cellular Digital Packet Data
ChinaSat	China Telecommunications Broadcast Satellite Corporation
CIDR	Classless Inter-Domain Routing
CRM	Customer Relationship Management
DAG	Data Acquisition and Generation
DANTE	Delivery of Advanced Network Technology to Europe
DSM	Data Stream Manager
EComm	Emergency Communications for Southwest British Columbia
EGP	Exterior Gateway Protocol
FIB	Forwarding Information Base
FIFO	First in First Out
FPGA	Field Programmable Gate Array
GVRD	Greater Vancouver Regional District
IANA	Internet Assigned Numbers Authority
IETF	Internet Engineering Task Force
IGP	Interior Gateway Protocol
IP	Internet Protocol
IPv4	Internet Protocol version 4
IPv6	Internet Protocol version 6
ISP	Internet Service Provider
LACNIC	Latin American and Caribbean Internet Address Registry

LIR	Local Internet Registry
MED	Multiple Exit Discriminator
MRT	Multi-threaded Routing Toolkit
NIR	National Internet Registry
NLRI	Network Layer Reachability Information
PCIx	Peripheral Component Interconnect Extended
PDF	Probability Density Function
PTT	Push to Talk
RIB	Routing Information Base
RIP	Routing Information Protocol
RIPE	Réseaux IP Européens
RIPE NCC	Réseaux IP Européens Network Coordination Centre
RIR	Regional Internet Registries
RTO	Retransmission Timeout
RTT	Round Trip Time
SARIMA	Seasonal Autoregressive Integrative Moving Average
TAP	Test Access Point
TCP	Transmission Control Protocol
TTL	Time to Live
UDP	User Datagram Protocol
VLAN	Virtual Local Area Network

1. Introduction

BCNET is British Columbia's advanced research and innovation network. It is a dedicated, high-speed, fiber-optic network that spans the province of British Columbia, Canada. Border Gateway Protocol (BGP) provides a set of mechanisms for supporting classless inter-domain routing, a path vector protocol which enables the use of additional address space within the network, in contrast to the current Internet Protocol (IP). It maintains a table of IP addresses or prefixes that define network reachability among Autonomous Systems (ASes). It supports any policy conforming to the “hop-by-hop” paradigm [1].Among the routing protocols, BGP is the only protocol that deals with a network of the size of the Internet and it may support multiple connections to distinct routing fields. BGP is the routing protocol that exchanges routing information across the Internet and makes it feasible for Internet Service Providers (ISPs) to connect to each other and for users to connect to more than one ISP. BGP speakers are the BGP routers. These routers communicate directly with each other and may be located within the same or different ASes. The most important function of a BGP speaker is to exchange network reachability information with other BGP systems. If the BGP speaker communicates with a router that is within the same Autonomous System, it is known as an Intra-Autonomous System. If the routers communicate with each other and are in different Autonomous Systems, then it is known as an Inter-Autonomous System.

Traffic measurements in operational networks help understand traffic characteristics, develop traffic models, and evaluate the performance of protocols and applications. BGP analysis may be grouped into three categories: a) measurement and modeling studies, b) studies of network-wide BGP dynamics, and c) troubleshooting and improving BGP [2]. This project falls in the category of measuring BGP.

It is difficult to collect real traffic from real deployed network. We had a great opportunity of collecting the traffic from BCNET for a period of 2 days in 2010, 2 weeks in 2011, and then a month and a half in 2011. Collecting traffic from BCNET is an ongoing project and we are continuing to collect the traffic until date. In this thesis, we collected traffic from BGP routing tables. The data extracted from BCNET were in American Standard Code for Information Interchange (ASCII) format. We used code written in C# to parse the ASCII format and then extract BGP message attributes.

1.1. Contributions

In this thesis, we describe the collection of data from BCNET using special purpose hardware. We cleaned, extracted, and then analyzed BGP traffic. We used Wireshark to analyze the data and compared the data collected between: December 20, 2010 to December 22, 2010; October 1, 2011 to October 10, 2011; and October 24, 2011 to December 22, 2011. We used C# code to parse the BGP data and extract numerical attributes. MATLAB was used to generate the graphs. BGP data extracted from BCNET contains only *update* and *keepalive* messages. There were no *notification* or *open* messages found in the collected data in December 2010, October 2011, November 2011, and December 2011. In the collected BGP traffic data, 88% of the messages were BGP *update* messages and the remaining were *keepalive* messages. There were few *keepalive* attributes to be analyzed. Hence, in this thesis, we focused our analysis on the *update* messages and their attributes..

It was difficult to extract any useful information and plot the graphs of different attributes using Wireshark. Therefore, it was important to have a tool that could parse the data and help in displaying the graphs. Hence, the C# code was written to extract useful data and plot the numerical attributes of the BGP messages.

The data collected from BCNET may be used to improve existing routing protocols such as BGP and eliminate the defects in BGP such as route flap damping (RFD) [3] and improve the values used for the minimal route arrival interval (MRAI) [4]. Route flap damping (RFD) is the pattern of repeated withdrawal and re-announcement of routes. MRAI is defined as the minimum time interval between sending two consecutive update messages for the same destination. The collected local BCNET data may be analyzed and compared with other publicly available datasets such as Route Views [5] and Réseaux IP Européens (RIPE) [6].

1.2. Thesis Outline

The organization of the thesis is as follows: Chapter 2 begins with a brief introduction to packet routing, an overview of BGP and BGP routing, and a description of Autonomous Systems. In Chapter 3, we discuss the process of data collection. The hardware used for BCNET data capture such as Endace card, Ninjabox 5000, and Netoptics Director 7400 are discussed in detail in this chapter. The BCNET network, BCNET traffic, and discussion on real time usage by BCNET members are also included in this Chapter. In Chapter 4, we describe a tool for viewing collected traffic. In this thesis, we chose a free source packet analyzer called Wireshark for this purpose. Various features such as input-output graphs, protocol hierarchy, and service response time are detailed in this Chapter. We analyze BGP attributes using MATLAB in Chapter 5 and TCP attributes in Chapter 6. Chapter 7 emphasizes the details of BGP *update* attributes and the traffic collected on three randomly selected dates in October, November, and December 2011. A short summary of experiences that we gained and the future work is addressed in Chapter 7. We conclude the thesis with Chapter 8. C# code for BGP data generation and MATLAB code are presented in the Appendices.

1.3. Related Work

Network traffic measurements are useful for network troubleshooting, workload characterization, and network performance evaluation. Collection and analysis of traffic has been an active research topic. Measuring and analyzing of network traces is important in order to understand traffic, illustrate traffic, and develop new traffic models, and improve network performance.

Ethernet LAN traffic is statistically self-similar and has serious implications for the design, control, and analysis of high-speed and cell-based networks [7]. Data were collected between August 1989 and February 1992 on several Ethernet LANs at the Bellcore Moristown Research and Engineering Center to analyze the self-similarity trend. The self-similarity of Ethernet LAN traffic is different from both conventional telephone traffic and packet traffic. The self-similarity is defined by the Hurst parameter. Four sets of traffic measurements of 20 and 40 consecutive hours of Ethernet traffic were considered. The degree of self-similarity increased as the utilization of the Ethernet increased and this was supported by self-similarity of the Ethernet traffic over a 24-hour period. Interest in self-similar traffic was first stirred by the measurements of Ethernet traffic at Bellcore [7].

In order to demonstrate that wireless data traffic also shows long-range dependent behavior it is important to study the impact of self-similarity on wireless data networks [8]. Modeling and simulation of Cellular Digital Packet Data (CDPD) [9] network was done to investigate the impact of traffic patterns on wireless data networks. The data were collected from Telus Mobility, a commercial service provider and its CDPD network located in downtown Vancouver. The traffic was collected from 14:56:37:56 to 15:24:46:88 on June 12, 1998. Long-range dependence is an important feature of self-similar traffic. The long-range dependent behavior in this case was different from the behaviour generated by traditional traffic models.

OPNET tool [10] was used to study the CDPD wireless networks. CDPD is a standard protocol stack developed for mobile data networks. While it is similar to Ethernet (IEEE 802.3), it differs from other protocols because it has wireless transmission medium and collision detection mechanism. The results in this case concluded that wireless data traffic had self-similar behavior, which was different from traditional traffic models. Queuing delays generated from authentic traffic traces were different from the queuing delay predicted by short-range dependent models. Hence, it is important to study real data from real deployed network.

Analysis of data from China Telecommunications Broadcast Satellite Corporation (ChinaSat) [11] dealt with analyzing the patterns and statistical properties of billing records and tcpdump traces [12]. ChinaSat provides Internet access through DirecPC to individual users, businesses, and Internet cafés in China. DirecPC is a satellite network deployed by Hughes Network Systems. The daily and weekly traffic patterns and effect of holidays were investigated using billing records. The volume of traffic on weekends was lower than working weekdays. Two months of billings records collected from DirecPC were analyzed. The downloaded traffic was larger than uploaded traffic. The collected traffic only contained IP packets because it is one of the most used network layer protocol. A large number of Routing Information Protocols (RIPs) were detected but they were not analyzed as they were not related to DirecPC traffic in ChinaSat network.

Wireshark and tcptrace were used to examine the traffic traces. Analysis of tcpdump traces revealed traffic anomalies such as packets with invalid TCP flag combinations, large number of connection closed, port scans, and anomalies in traffic volume. Wavelets decomposition of data traffic may be used to identify traffic volume anomalies. The download and upload traffic traces were very regular in both daily and weekly cycles. The daily maxima occurred at 11 AM, 3 PM, and 7 PM while minimum occurred at 7 AM. Analysis of tcpdump traces indicated that the major data transfers were due to TCP. Some traffic anomalies were also detected. The performance of commercial network may be improved using this analysis.

Analysis of public safety traffic dealt with distribution of call inter-arrival and call holding times for multicast voice traffic on a transmission trunked mobile radio system [13]. The traffic was collected from a deployed network called E-Comm [14]. It is an emergency communications center that provides emergency services such as police, fire, rescue, and ambulance in Greater Vancouver Regional District (GVRD) in Southwest BC, Canada. Each call had a push-to-talk (PTT) event to activate the radio transmitter and it ended with the release of the PTT key. The channel at each repeater site is released when the PTT key is released. If there are no channels available, the PTT request is located in a queue and served on a first-in–first-out (FIFO). The highest priorities are reserved for emergencies.

Analysis of several busy periods on weekdays was done and two days of traffic from the Vancouver system were examined in detail. All confidential information that could identify specific user group was removed. The traffic analysis consisted of traces from the Vancouver system that handled bulky traffic volumes and had capacity so that blockage and queueing did not occur. The traffic patterns were observed for September, October, and November 2001 over a seven-week period.

The traffic in a single coverage system was only considered. This removal of mobility reduced the traffic to a function of a call-arrival process and a call-holding process. The long-range dependence was investigated for the call inter-arrival and call holding times by estimating the Hurst parameter. The analysis indicated that the call holding times had a lognormal distribution. The analysis included the distributions of call inter-arrival times, call holding times, probability density functions (PDF), and cumulative distribution functions (CDF).

Analysis and mining of traffic data helps in determining traffic distribution, to study user behavior patterns, and to predict future network traffic [15]. Analysis may improve the network resources usage and quality of service. The data were collected from E-Comm network. K-means algorithm was used to cluster and classify user groups based on their calling patterns. The databases contained log tables recording all the network activities. The data was analyzed from March 2003 to May 2003. All the fields in the databases were not required for analysis. Therefore, the data were cleaned and the fields of interest were taken into consideration. After the cleaning process, the records were reduced to 19% of their original records.

The K-means algorithm was used to cluster the users according to their calling patterns. The users were clustered in such a way that they had high similarities if they were in the same cluster and very few similarities if they were in other clusters. The number of clusters and the similarity function are two parameters used in determining the clusters. Three main clusters were recognized in the traffic analyzed. The first cluster had 17 talk groups (It represented the busiest calling group.), and the second cluster had 31 talk groups (It represented the callers with medium network usage.), and the third cluster had 569 talk groups (The callers made approximately 16 calls per hour.).

Traffic was also predicted based on aggregate traffic using the autoregressive integrative moving average (ARIMA) and seasonal ARIMA (SARIMA) models. The ARIMA model was able to predict the uploaded traffic but not the downloaded traffic, due to the traffic dynamics. E-Comm network consisted of both daily (24-hour) and weekly (168-hour) cyclic patterns. The daily model assumes that traffic is relatively constant for a weekday while the weekly model assumes there are variations in the traffic during the week. Predicting traffic of Thursday based on Wednesday’s data is not as accurate as predicting Thursday’s traffic based on traffic on previous Thursdays. The prediction of traffic based on aggregate traffic assumed that the number of network users is constant and they have a steady behavior pattern. The cluster based technique is used to predict the network traffic in case the network expands. Another advantage of using cluster-based prediction is that the number of users may vary but they may be grouped into one of the existing clusters.

Power-laws might be used to estimate important parameters such as the performance and analysis of protocols [16]. The information in this case was collected from BGP routing tables. Three power-laws of the Internet topology were observed. Many fascinating graph parameter relationships were observed. Power-laws helped understanding different graph metrics. The parameters of the Internet may be predicted using hypothesis and assumptions. They may be used to illustrate and differentiate Internet graphs. Hence, power-laws are another way to study the Internet topology. Power-laws have theoretical as well as practical applications.

The analysis of spectral properties of Internet topology is important [17]. The Internet is a complex network and in order to understand it in depth we need to study its spectral properties. Datasets from Route Views and RIPE collected from BGP routing tables indicated the existence of power-laws. The Internet has been growing exponentially over the years but the power-laws have not much changed. Spectral analysis was employed to analyze Route View and RIPE so that clustering of AS nodes could be studied. Eigenvalues are an important feature in order to study spectral properties of the Internet. They demonstrate power-law properties. Various graphical properties from Route Views and RIPE in 2003 and 2008 were examined. The Route Views and RIPE BGP routing tables were collected from different ASes located in different countries. Most of the Route Views ASes were in North America, whereas RIPE consisted of ASes in Europe. RIPE datasets were collected from 16 different locations. The datasets that were analyzed were collected at 00:00 am on July 31, 2003 and 00:00 am on July 31, 2008.

RIPE datasets showed properties similar to Route Views observed power-laws. The second smallest and largest eigenvalues and their eigenvectors for both Route Views and RIPE were examined. Route Views 2003 and RIPE 2003 datasets showed similar clustering patterns and so did the Route Views 2008 and RIPE 2008 datasets. Even though the Route Views and RIPE datasets were collected from different countries, similar power-laws were present in both. While, many properties in Internet topology have not changed over the years, the spectral analysis showed changes in both connectivity and clustering of AS nodes.

Analyzing network traces is important in order to understand the Internet. However, obtaining the traces is a difficult task [18]. Privacy issues and the operation of the network are some of the issues concerned with obtaining the traces. However, many traces are available to researchers on the web. A Google-based profiling tool known as “unconstrained endpoint profiling” (UEP) approach was used for this purpose. The method used information about the endpoints that was publicly-available on the web. This approach may be used even when no packet traces are available or when network traces are available, and sampled flow-level traces are available. Different regions in Asia, South America, North America, and Europe were investigated to understand what protocols and applications people use and which are the sites they access. The traffic categories that were compared were P2P, chat, gaming, and browsing. The best way to do so is by analyzing the network traces. Sampled data analysis creates problems in anomaly detection algorithms and traffic classification tools. The endpoint profiling approach may be used to predict application and protocol usage trends. It surpasses modern classification tools when packet traces are available, and preserves high classification means. It was the first-of-its-kind endpoint analysis that revealed similarities and differences among these regions from where different traces were collected.

Some of the other tools related to BGP data analysis are BGP monitoring system (BGPmon) [19], BGP Data Analysis Project (BDAP) [20], and BGP-Inspect [21]. The main aim of these projects is to collect BGP data, to monitor BGP updates and routing tables, and to detect anomalies in the data. These tools are mainly designed to detect the instabilities in the Internet routes.

2. Border Gateway Protocol

The standard inter-domain routing protocol in today's Internet is Border Gateway Protocol (BGP), defined in the RFC 1771. BGP is categorized as a path vector protocol, a variant of distance vector protocol. It exchanges network reachability information among BGP systems. It distributes route path information to peers and sends update messages as routing tables change. The size of BGP tables has exponentially increased since 1994, as shown in Figure 1, implying that timely analysis of BGP is important [22].

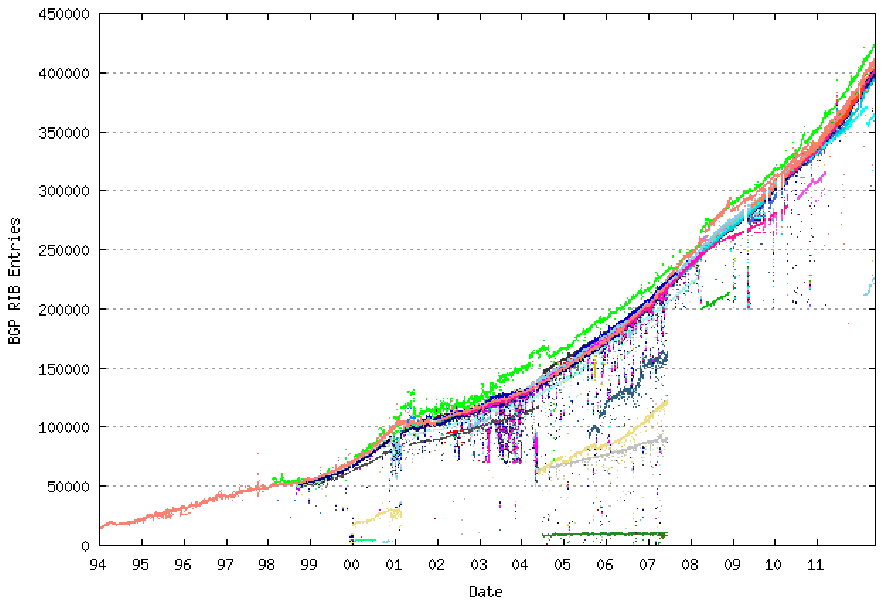


Figure 1. Growth of the BGP Table - 1994 to Present [22].

Analyzing the BGP routing tables is important. As the Internet is growing, the number of entries in the BGP routing tables is increasing. Hence, the Internet address space and capability of routing system is limited. Therefore, it is important to study the in routing table entries [23]. The graph illustrates that the number of BGP routing table or Routing Information Base (RIB) entries have increased from 50,000 in 1994 to approximately 450,000 entries in 2011. Various reported research projects [3] – [18]

imply that traffic analysis is important for improving network performance, evaluating protocols, detecting the anomalies.

2.1. Routing

The process of choosing paths through which network traffic is sent is known as routing. It is performed in many different types of networks, such as electronic data networks (Internet), telephone networks (circuit switching), and transportation networks.

There are five types of routing schemes based on their delivery method:

- Unicast: A message is delivered to a single specific node.
- Broadcast: A message is delivered to all nodes in the network.
- Multicast: A message is delivered to the nodes that are interested in receiving the message.
- Anycast: A message is delivered to the node nearest to the source from a group of nodes.
- Geocast: A message is delivered to a particular geographical area.

2.2. BGP Routing

BGP maintains routing tables, transmits routing updates, and bases routing decisions on routing metrics. Each BGP router preserves a routing table that lists all viable paths to a particular network. Routing information received from peer routers is maintained until an incremental update is received and the router does not change the routing table.

BGP devices exchange routing information after preliminary data exchange and incremental updates. When a router first connects to the network, all BGP routers exchange their entire BGP routing tables. When the routing table changes, the router sends the portion of its routing table that has changed to its neighbors. BGP routing updates announce only the optimal path to a network and BGP routers do not send regularly scheduled routing updates.

Every router maintains two groups of routes: Routing Information Base (RIB) and Forwarding Information Base (FIB). The RIB consists of all the announced routes from the neighbor routers, whereas the FIB contains the best route to each destination, calculated by the route-selection algorithm from the RIB. Since BGP does not support multi-path routing, if several routes are considered as the best paths, a rule is applied to select the one to be placed in FIB.

BGP uses a single routing metric to determine the best path to a given network. The routing metric consists of a random integer that specifies the degree of preference of a particular link. The BGP metric is usually assigned to each link by the network supervisor. The value assigned to a link may be based on different criteria, including the number of autonomous systems through which the path passes, stability, speed, delay, or cost.

2.3. BGP Overview

Border Gateway Protocol (BGP) is a *de facto* Inter-Autonomous System (AS) [24] routing protocol. An Autonomous System comprises groups of routers that are administrated by a single administrator and use the Interior Gateway Protocol (IGP). Each AS is responsible for carrying traffic to and from a set of customer IP addresses. The Autonomous System numbers are used by various routing protocols. They are assigned to the regional registries by the Internet Assigned Numbers Authority (IANA).

BGP operates over a Transmission Control Protocol (TCP) as a transport protocol (port number 179). TCP has an advantage over User Datagram Protocol (UDP) connections: BGP does not need to implement fragmentation, retransmission, acknowledgment, and sequencing. BGP has the capability to support Classless Inter-Domain Routing (CIDR) in order to reduce the size of the Internet routing tables. CIDR allows routers to group routes together in order to minimize the number of routing information carried by the core routers, which makes it a dominant Internet routing protocol and allows the aggregation of routers. CIDR is also known as supernet as it effectively allows multiple subnets to be grouped together for network routing. Internet Protocol version 6 (IPv6) uses CIDR routing technology and CIDR notation in the same way as Internet Protocol version 4 (IPv4). IPv6 was designed for fully classless addressing. In CIDR, all Internet blocks can be of random size and classless addressing uses a variable number of bits for the network and host portions of the address.

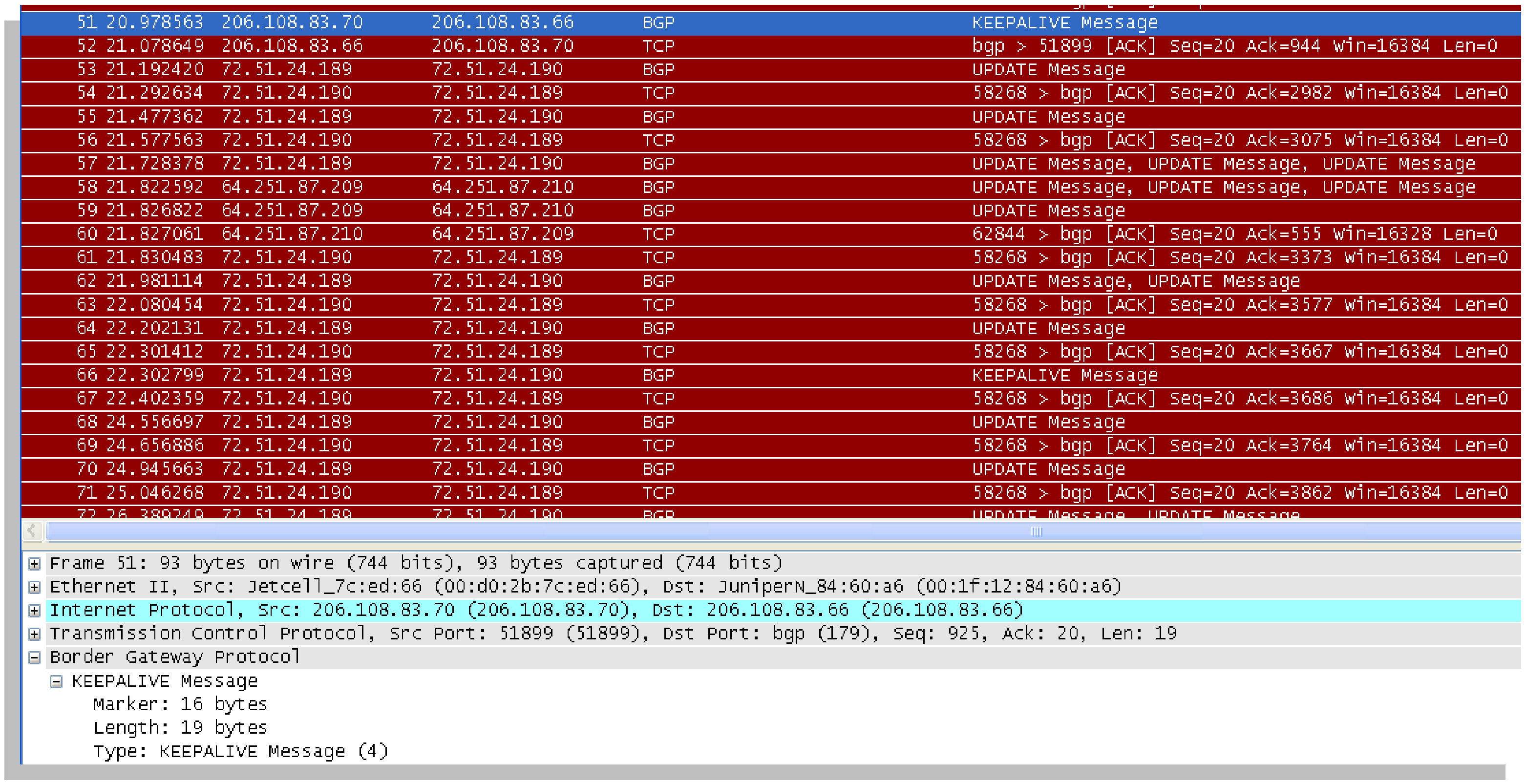
BGP is a path-vector protocol that is commonly used for exchanging external AS routing information and operates at the level of address blocks or AS prefixes. Each AS prefix consists of a 32-bit address and a mask length. For example, 192.0.2.0/24 consists of addresses from 192.0.2.0 to 192.0.2.255 [25].

BGP speakers that participate in a BGP session are called neighbors or peers. The main function of BGP is to exchange reachability information among BGP systems and this information is based on a set of metrics: policy decision, the shortest AS-PATH, and the closest NEXT-HOP router.

BGP routers exchange routing information using four types of messages [12]:

- *Open*: After a TCP connection is established, each BGP peer sends an *open* message to open an initial connection. This is the first message sent between peers after the TCP connection is established.
- *Update*: The *update* message is used to transfer and update routing information between BGP peers. As the routing table changes, incremental updates are sent to peers using an *update* message. The *update* information allows routers to construct network topology view that describes the relationships between various ASes.
- *Notification*: The *notification* message is sent to all connected neighbors in case of errors or unusual conditions. If a connection between the connected routers has an error, a *notification* message announces the error and closes the active connection between the routers.
- *Keepalive*: The *keepalive* message assists the BGP router to determine whether or not the peers are reachable. BGP router sends a *keepalive* message between peers occasionally in order to ensure the active connection between them.

A snippet of captured data showing *update* and *keepalive* messages is shown in Figure 2. We did not encounter any *notification* and *open* messages in traffic collected from BCNET. Therefore, our focus was to analyze BGP *update* and *keepalive* message attributes.



```
 51 20.978563  206.108.83.70    206.108.83.66    BGP    KEEPALIVE Message
 52 21.078649  206.108.83.66    206.108.83.70    TCP    bgp > 51899 [ACK] Seq=20 Ack=944 Win=16384 Len=0
 53 21.192420  72.51.24.189     72.51.24.190     BGP    UPDATE Message
 54 21.292634  72.51.24.190     72.51.24.189     TCP    58268 > bgp [ACK] Seq=20 Ack=2982 Win=16384 Len=0
 55 21.477362  72.51.24.189     72.51.24.190     BGP    UPDATE Message
 56 21.577563  72.51.24.190     72.51.24.189     TCP    58268 > bgp [ACK] Seq=20 Ack=3075 Win=16384 Len=0
 57 21.728378  72.51.24.189     72.51.24.190     BGP    UPDATE Message, UPDATE Message, UPDATE Message
 58 21.822592  64.251.87.209    64.251.87.210    BGP    UPDATE Message, UPDATE Message, UPDATE Message
 59 21.826822  64.251.87.209    64.251.87.210    BGP    UPDATE Message
 60 21.827061  64.251.87.210    64.251.87.209    TCP    62844 > bgp [ACK] Seq=20 Ack=555 Win=16328 Len=0
 61 21.830483  72.51.24.190     72.51.24.189     TCP    58268 > bgp [ACK] Seq=20 Ack=3373 Win=16384 Len=0
 62 21.981114  72.51.24.189     72.51.24.190     BGP    UPDATE Message, UPDATE Message
 63 22.080454  72.51.24.190     72.51.24.189     TCP    58268 > bgp [ACK] Seq=20 Ack=3577 Win=16384 Len=0
 64 22.202131  72.51.24.189     72.51.24.190     BGP    UPDATE Message
 65 22.301412  72.51.24.190     72.51.24.189     TCP    58268 > bgp [ACK] Seq=20 Ack=3667 Win=16384 Len=0
 66 22.302799  72.51.24.189     72.51.24.190     BGP    KEEPALIVE Message
 67 22.402359  72.51.24.190     72.51.24.189     TCP    58268 > bgp [ACK] Seq=20 Ack=3686 Win=16384 Len=0
 68 24.556697  72.51.24.189     72.51.24.190     BGP    UPDATE Message
 69 24.656886  72.51.24.190     72.51.24.189     TCP    58268 > bgp [ACK] Seq=20 Ack=3764 Win=16384 Len=0
 70 24.945663  72.51.24.189     72.51.24.190     BGP    UPDATE Message
 71 25.046268  72.51.24.190     72.51.24.189     TCP    58268 > bgp [ACK] Seq=20 Ack=3862 Win=16384 Len=0
 72 26.389249  72.51.24.189     72.51.24.190     BGP    UPDATE Message, UPDATE Message

⊞ Frame 51: 93 bytes on wire (744 bits), 93 bytes captured (744 bits)
⊞ Ethernet II, Src: Jetcell_7c:ed:66 (00:d0:2b:7c:ed:66), Dst: JuniperN_84:60:a6 (00:1f:12:84:60:a6)
⊞ Internet Protocol, Src: 206.108.83.70 (206.108.83.70), Dst: 206.108.83.66 (206.108.83.66)
⊞ Transmission Control Protocol, Src Port: 51899 (51899), Dst Port: bgp (179), Seq: 925, Ack: 20, Len: 19
⊟ Border Gateway Protocol
  ⊟ KEEPALIVE Message
      Marker: 16 bytes
      Length: 19 bytes
      Type: KEEPALIVE Message (4)
```

Figure 2. Snippet of Captured Data Showing Update and Keepalive Messages using Wireshark.

2.4. Autonomous Systems (ASes)

The Internet is composed of a set of administrated networks, known as Autonomous Systems (ASes). An AS is an association that manages its private networks and provides Internet access to its hosts by being connected through other ASes. Additionally, each AS has one or more unique IP prefixes, i.e., a range of IP addresses, from which it assigns IP addresses to its hosts. ASes run an inter-domain routing protocol called Border Gateway Routing Protocol (BGP) to exchange reachability and dynamic information (link failures and availability) regarding their prefixes.

The AS is a network or a group of networks with a common routing policy. For example, an AS may consist of a university network, a business enterprise, or a corporation network. A network within an AS uses a common IGP to route packets. BGP information at each AS router is kept consistent by BGP *update* messages received from BGP routers belonging to other ASes.

Until 2006, the AS numbers were defined as 16-bit integers and IANA allowed a maximum of 65,536 allocations. Since November 2006, IANA has extended the ASN field from 16 bits to 32 bits and hence the pool size has increased to 4,294,967,296 values. The IANA has designated AS numbers 64,512 through 65,534 to be used for private purposes. The ASNs 0, 59,392–64,511, and 65,535 are reserved and should not be used in any routing environment. The number of unique autonomous networks in the routing system of the Internet exceeded 5,000 in 1999, 30,000 in late 2008, and 35,000 in 2010.

. Each AS is identified by a unique number known as Autonomous System Number (ASN) assigned by the Internet Assigned Number Authority (IANA). According to the global policy, IANA allocates the Internet Protocol addresses from the pool of unallocated addresses to the Regional Internet Registries (RIR). An AS consists of a range of IP addresses; the Internet Service Providers (ISPs) assign these IP addresses to its users [26].

The Internet topology may be analyzed at router level and the Autonomous System level. At inter-domain level or Autonomous System level, each AS domain is represented as a node. The AS graph represents the connections between ASes and these AS connections are the links between two nodes. Each AS is represented by an ASN that ranges from 0 to 65,535.

The allocated pool of numbers is managed by the RIRs. The AS number blocks allocated by IANA [25] are shown in Figure 3. 1,042 numbers are reserved for IETF and 3,056 numbers are for IANA pool. The sum of all these numbers sum is 65,536. ISPs obtain IP addresses from a Local Internet Registry (LIR), National Internet Registry (NIR), or from their RIR:

- African Regional Internet Registry (AfriNIC), Africa region.
- Asia Pacific Network Information Centre (APNIC), Asia Pacific region.
- American Registry for Internet Numbers (ARIN), North America region.
- Latin American and Caribbean Internet Address Registry (LACNIC), Latin America and Caribbean Islands region.
- Réseaux IP Européens Network Coordination Centre (RIPE NCC), Europe, Middle East, and Central Asia region.

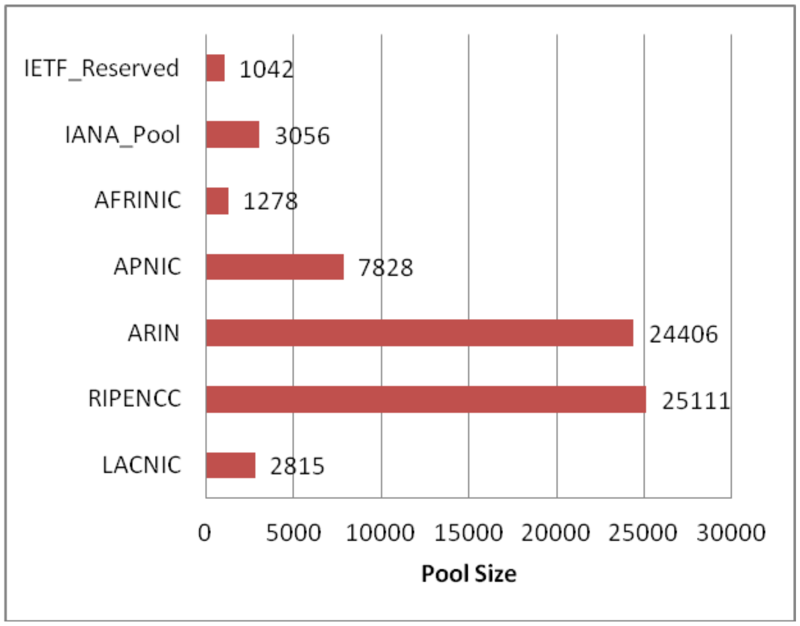


Figure 3. AS Pool of Numbers Allocated by IANA.

3. Data Collection

The traffic data analyzed in this thesis were obtained from BCNET [27]. In this Chapter, we introduce the architecture of BCNET and the special-purpose hardware that was used for the traffic capture.

3.1. BCNET

The BCNET network facilitates high-definition videoconferencing, remote research, virtual laboratories, distributed computing, distant learning, and large-scale data transfers. The BCNET network is a high-speed fiber optic research network used to transmit telephone signals, Internet communication, and cable television signals. This network has been primarily installed for long-distance applications, where it may be used to its full transmission capacity. The major Internet service providers in Vancouver, Prince George, Victoria, Kelowna, and Kamloops are Bell, Shaw, and Telus.

Network interconnections are accommodated by BCNET transit exchanges that implement peering between associates and access data exchanges with local peering and multi-homing services. The peering requires an exchange of routing information and physical interconnection of networks through BGP routing protocol. The BCNET offsets the increased Internet transit cost and improves network performance. Local peering services are used to connect an organization at the BCNET Transit Exchange to move data effectively and efficiently. Multi-homing establishes multiple Internet connections for an organization at any of the five BCNET Transit Exchange locations.

3.2. BCNET Traffic

BCNET is associated with the network alliance Canada's Advanced Research and Innovation Network (CANARIE), which links Canada to the United States through

Internet and to Europe through Delivery of Advanced Network Technology to Europe (DANTE). The BCNET traffic map, shown in Figure 4, displays the real-time network usage by BCNET associates. Vancouver is connected to Prince George, Victoria, and Kelowna through Kamloops. All these cities are then connected to their respective transits. The arrows in the map show the traffic bound for CANARIE, the commercial Internet (Transits), and peering traffic at the Seattle Internet Exchange (Seattle IX). The numbers show the BCNET traffic load being exchanged between cities and their transits.

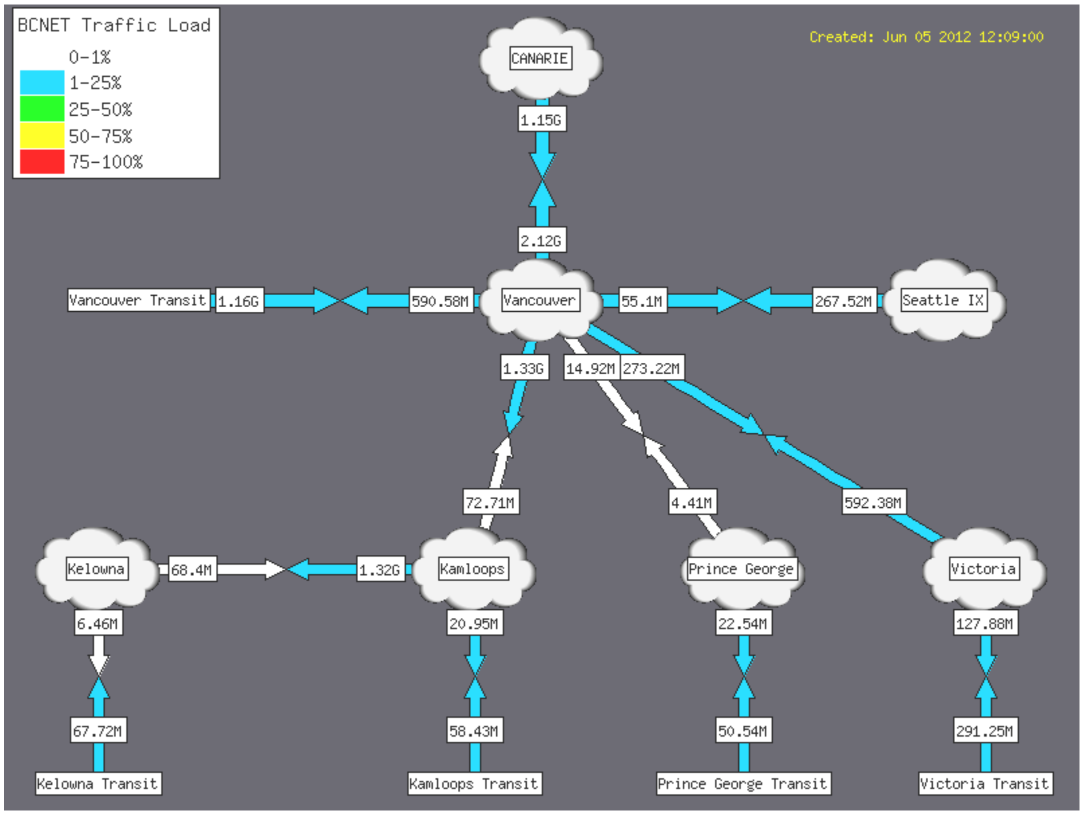


Figure 4. Real Time Network Usage by BCNET Members, Collected on June 5, 2012 [28].

3.3. BCNET Packet Capture

BCNET network is one of the most advanced, high-speed fiber-optic research networks in the world. As the name suggests, it is a network in British Columbia (BC), Canada that extends over 1,400 kilometers. It connects Calgary to Kamloops and Kelowna, and extends through Vancouver to Prince George and Victoria. It offers up to

72 wavelengths of capacity at 10 Gbps. The network map of BCNET is shown in Figure 5.

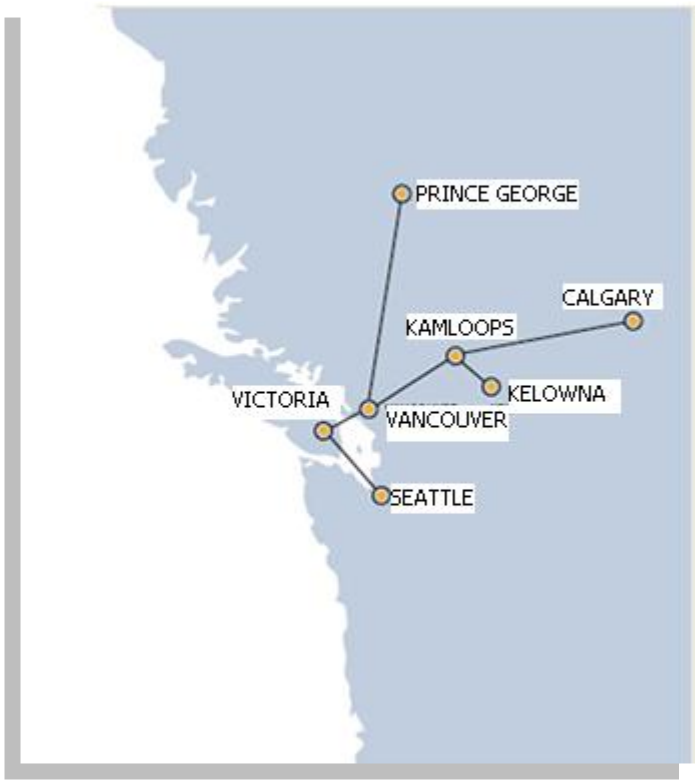


Figure 5. BCNET, the British Columbia's Advanced Network.

BCNET is the hub of advanced telecommunication networks in British Columbia, Canada that offers services to research and higher education institutions. This advanced network offers unconstrained bandwidth to research and innovation centers making it suitable to address the unique requirements of researchers [27]. It is used for collaboration among researchers across institutions residing in British Columbia.

A physical overview map of the BCNET is shown in Figure 6. BCNET transits have two service providers with 10 Gbps network links and one service provider with 1 Gbps network link. During the data collection from December 20, 2010 to December 22, 2010 two 10 Gbps transit service providers were Shaw Cable Systems and Telus Advanced Communications and 1 Gbps transit service provider was Peer 1 Networks Inc. These links are connected via two routers: BCNET Router 1 and BCNET Router 2. They are placed in two separate physical rooms. Router 1 is connected via 10-Gig link services while Router 2 connects both 10 Gbps and 1 Gbps providers. BCNET transit exchange connects the participants and provides local peering and multi-homing services to open and private data exchanges.

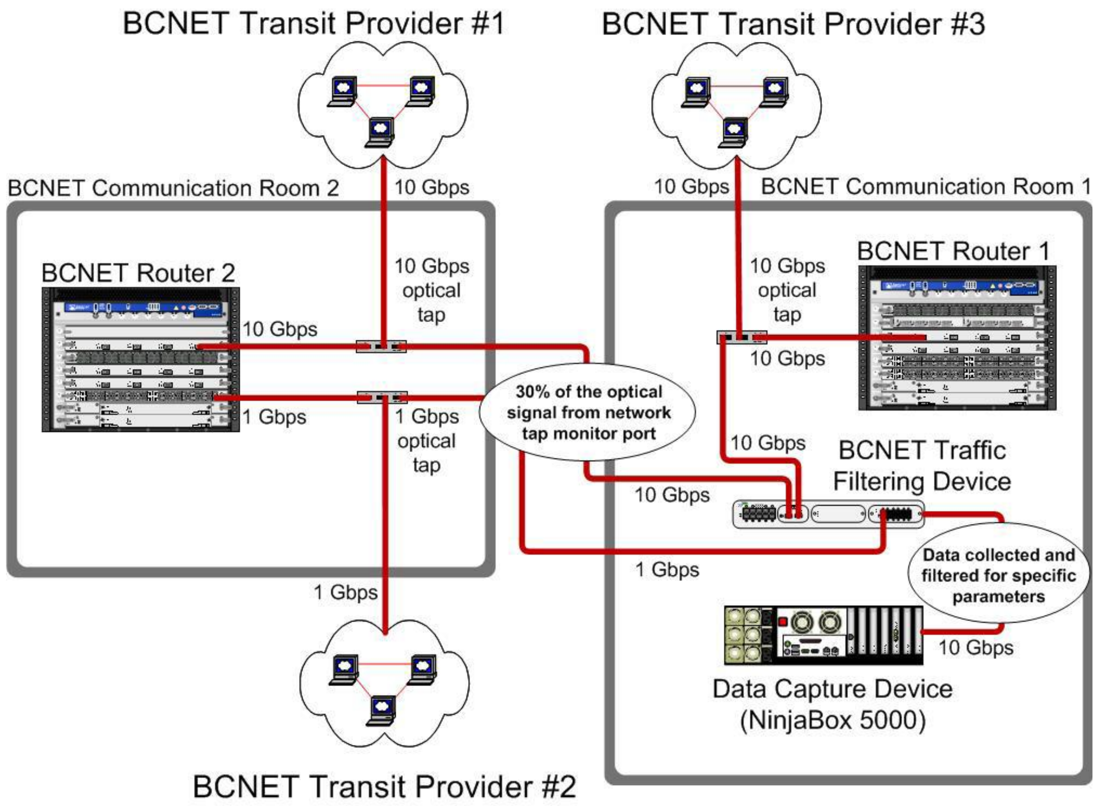

BCNET Transit Provider #1
BCNET Transit Provider #3
BCNET Communication Room 2
10 Gbps
10 Gbps
BCNET Communication Room 1
BCNET Router 2
10 Gbps optical tap
10 Gbps optical tap
BCNET Router 1
10 Gbps
10 Gbps
1 Gbps
1 Gbps optical tap
30% of the optical signal from network tap monitor port
10 Gbps
BCNET Traffic Filtering Device
10 Gbps
1 Gbps
Data collected and filtered for specific parameters
1 Gbps
10 Gbps
Data Capture Device (NinjaBox 5000)
BCNET Transit Provider #2

Figure 6. Physical Overview of the BCNET Packet Capture [27].

The optical Test Access Point (TAP) adjacent to BCNET routers in both communication rooms shown in Figure 6 splits the signal into two distinct paths. The signals splitting ratio from TAP may be modified. In the BCNET transit exchange, 30% of the optical signal is directed to a Traffic Filtering Device while the remaining 70% is sent to routers for processing.

The Data Capture Device (NinjaBox 5000) collects real-time data from the BCNET Traffic Filtering Device, as shown in Figure 6. NinjaBox 5000 relies on the Linux operating system to capture data at line rates using traditionally made network monitoring. The NinjaBox 5000 comes preconfigured with Endace DAG monitoring interface technology. Conventional capture systems cannot capture data at high rates due to the overhead in processing the captured packets, which results in lost packets.

The Net Optics Director 7400 is the BCNET Traffic Filtering Device and is used for traffic filtering. All three BCNET service provider links are connected to the device, as shown in Figure 6. Captured data are filtered for a specific set of parameters by the filtering device and then directed to the NinjaBox 5000. The filtering device operates as a data-monitoring switch [29]. It directs traffic to monitoring tools such as NinjaBox 5000. The Net Optics Director application diagram is shown in Figure 7.

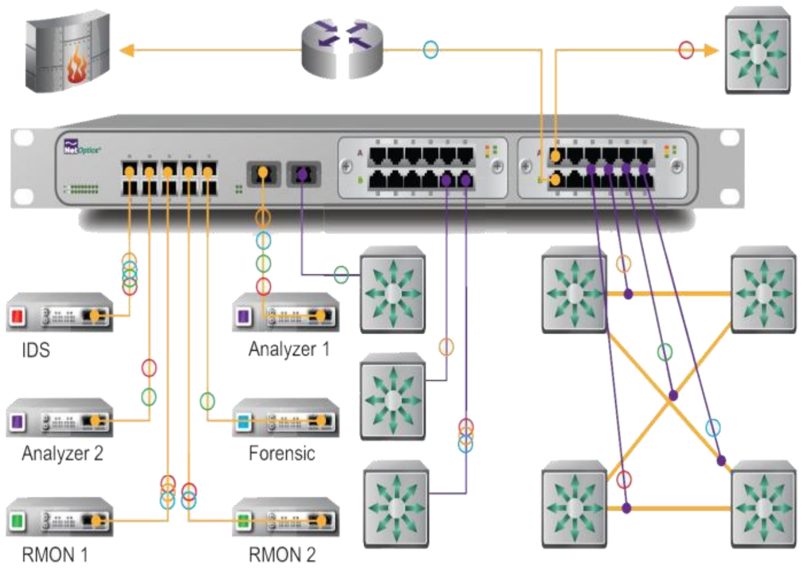


Figure 7. ***Net Optics Director Application Diagram [29].***

A data-monitoring switch (Director) provides access to traffic from network links. It provides functionalities that include monitoring traffic from multiple links, regenerating traffic to multiple tools, pre-filtering traffic to offload tools, and directing traffic according to one-to-one and many-to-many port mappings. It assists organizations to use their monitoring tools efficiently, to centralize traffic monitoring functions, and to share tools and traffic access between groups [30].

The Endace Data Acquisition and Generation (DAG) 5.2X card, shown in Figure 8, resides inside the NinjaBox 5000. It captures and transmits traffic and has time-stamping capability. Resolution of the Endace DAG time stamp is 10 ns. In contrast, software-based captures such as Wireshark support only 1 μs resolution [31]. The fine granularity of hardware-based captures ensures credibility, accuracy, and reliability of measured BGP traffic and its subsequent analysis, characterization, and modeling [32], [33].

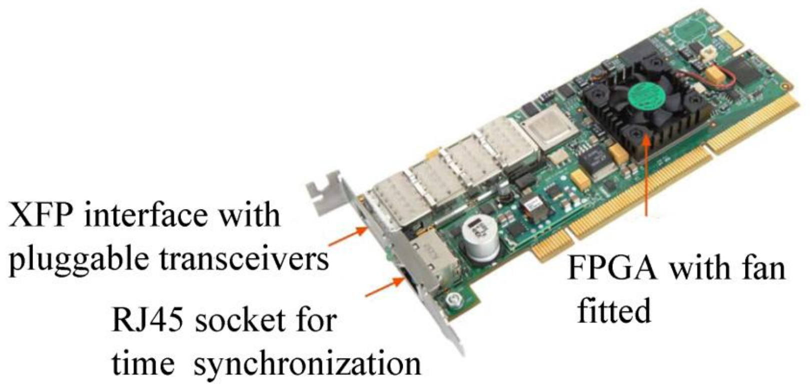

XFP interface with pluggable transceivers
RJ45 socket for time synchronization
FPGA with fan fitted

Figure 8. ENDACE Card is used for Network Monitoring and Analysis [34].

Endace specializes in high-performance network monitoring and analysis. “DAG” is a technology that combines hardware design using field programmable gate array (FPGA) technology and software design based on a programmable chip [34]. It also supports the Data Stream Manager (DSM) feature that allows discarding or routing packets to a particular stream based on the packet contents, physical port, and the output of load-balancing algorithms.

The DAG 5.2X card is a single-port Peripheral Component Interconnect Extended (PCIx) card. The card provides capture and transmission of full duplex optical 10 Gbps Ethernet network data. It is capable of capturing, on average, Ethernet traffic of 6.9 Gbps.

4. Viewing the Collected Traffic

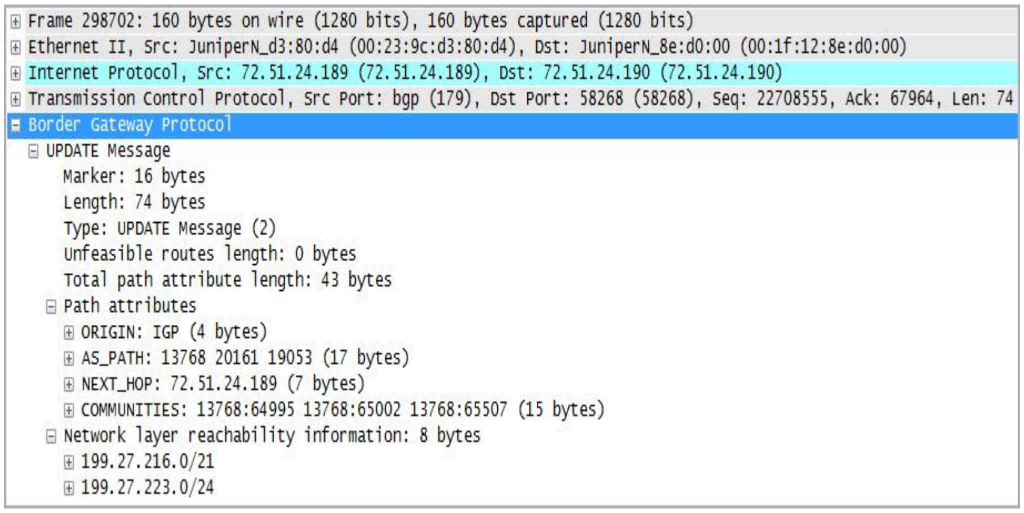
A tool was needed for tracking down networking problems and to learn more about networking. For this purpose, Ethereal was released in July 1998. In 2006, Ethereal emerged under a new name, Wireshark. It is one of the best publicly available packet analyzers available today.

4.1. Wireshark – A Packet Analyzer

Wireshark is an open-source packet analyzer that captures network packet data from a network interface and displays packets with detailed protocol information [31]. It is a measuring tool used to examine the contents of a network cable. Wireshark provides comprehensive statistics such as a summary of traffic collected, input/output graphs, protocol hierarchy, and endpoints.

A view of the traffic collected using Wireshark is shown in Figure 9. It illustrates the protocol structure for a randomly selected BGP *update* message, which contains path attributes for the advertised Network Layer Reachability Information (NLRI). It opens and saves captured packet data, imports and exports packet data from and to other capture programs, filters and searches packets based on various criteria, colorizes packet display based on filters, and creates various statistics.

Figure 9 illustrates frame number 298702 and the number of bytes captured on this frame. As messages originate from multiple protocols, the frame shows Ethernet protocol source and destination address, the source and destination addresses of IP, source and destinations port numbers for TCP, and details of BGP. The *update* message has a marker of 16 bytes and length of 74 bytes.



```
⊞ Frame 298702: 160 bytes on wire (1280 bits), 160 bytes captured (1280 bits)
⊞ Ethernet II, Src: JuniperN_d3:80:d4 (00:23:9c:d3:80:d4), Dst: JuniperN_8e:d0:00 (00:1f:12:8e:d0:00)
⊞ Internet Protocol, Src: 72.51.24.189 (72.51.24.189), Dst: 72.51.24.190 (72.51.24.190)
⊞ Transmission Control Protocol, Src Port: bgp (179), Dst Port: 58268 (58268), Seq: 22708555, Ack: 67964, Len: 74
⊟ Border Gateway Protocol
  ⊟ UPDATE Message
      Marker: 16 bytes
      Length: 74 bytes
      Type: UPDATE Message (2)
      Unfeasible routes length: 0 bytes
      Total path attribute length: 43 bytes
    ⊟ Path attributes
      ⊞ ORIGIN: IGP (4 bytes)
      ⊞ AS_PATH: 13768 20161 19053 (17 bytes)
      ⊞ NEXT_HOP: 72.51.24.189 (7 bytes)
      ⊞ COMMUNITIES: 13768:64995 13768:65002 13768:65507 (15 bytes)
    ⊟ Network layer reachability information: 8 bytes
      ⊞ 199.27.216.0/21
      ⊞ 199.27.223.0/24
```

Figure 9. Wireshark View of the Traffic Collected.

Type 2 indicates that this message is an *update* message, type 1 indicates *open* message, type 3 indicates *notification* message, and type 4 indicates *keepalive* message. IGP is assigned to the origin attribute, AS_path attribute has a length of 17 bytes, next_hop is composed of 7 bytes, and NLRI has 8 bytes. The BGP messages and their types are shown in Table 1.

Table 1. Different types of BGP messages.

Type	Message
1	Open
2	Update
3	Notification
4	Keepalive

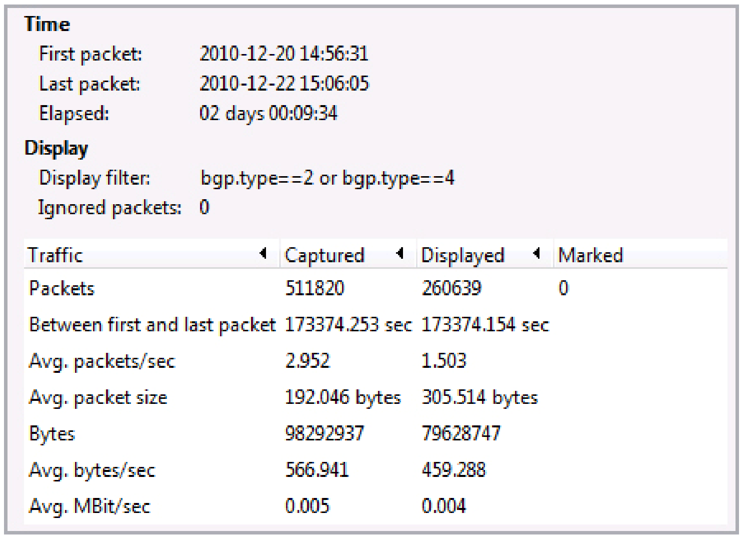
4.2. Analysis of BCNET Traffic Using Wireshark

The features defined below in this subsection are for the data collected from December 20, 2010 to December 22, 2010.

4.2.1. BCNET Traffic Summary

A summary of BCNET traffic collected is shown in Figure 10. The timestamps show when the first and the last packets were collected. There were 511,820 packets collected over the period of 48 hours between December 20, 2010 and December 22,

2010. The figure illustrates the time when the first and the last packet were received, the total time between the first and the last packet, the average number of packets captured, and the total number of bytes captured. An example of summary statistics for a specific filter for *update* and *keepalive* messages is shown in the displayed column.



Time

First packet:	2010-12-20 14:56:31
Last packet:	2010-12-22 15:06:05
Elapsed:	02 days 00:09:34

Display

Display filter:	bgp.type==2 or bgp.type==4
Ignored packets:	0

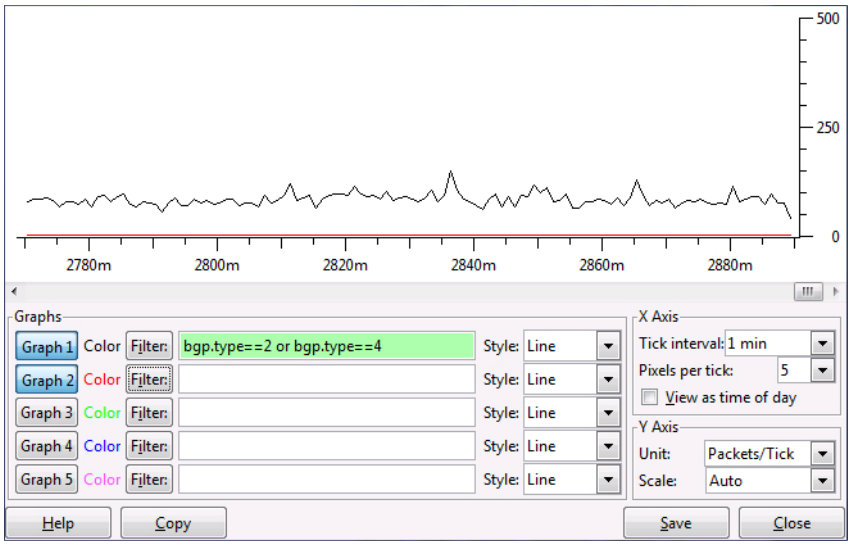
Traffic	Captured	Displayed	Marked
Packets	511820	260639	0
Between first and last packet	173374.253 sec	173374.154 sec	
Avg. packets/sec	2.952	1.503	
Avg. packet size	192.046 bytes	305.514 bytes	
Bytes	98292937	79628747	
Avg. bytes/sec	566.941	459.288	
Avg. MBit/sec	0.005	0.004	

Figure 10. Summary of BCNET Traffic Collected over a Period of 48 hours.

4.2.2. BCNET Traffic Input-Output Graphs

The traffic input-output graphs define up to five filters. The number of samples is limited to 100,000. A sample graph for two filters bgp.type == 2 (*update* message) and bgp.type == 4 (*keepalive* message) is shown in Figure 11.

The tick interval for x-axis may be chosen to be 0.001 s, 0.01 s, 0.1 s, 1 s, 10 s, 1 min or 10 min and the pixels/tick may be 1, 2, 5, or 10. In this figure the tick interval for the x-axis: = 1 s and 5 pixels/tick. The unit for y-axis may be packets/tick, bytes/tick, or bits/tick and the scale may be auto or logarithmic. We chose the y-axis: unit = packets/tick and scale = 10***.***



Graphs

Graph 1	Color	Filter: bgp.type==2 or bgp.type==4	Style:	Line
Graph 2	Color	Filter:	Style:	Line
Graph 3	Color	Filter:	Style:	Line
Graph 4	Color	Filter:	Style:	Line
Graph 5	Color	Filter:	Style:	Line

X Axis
Tick interval: 1 min
Pixels per tick: 5
View as time of day

Y Axis
Unit: Packets/Tick
Scale: Auto

Figure 11. ***Input-Output Graph of the Packets Captured. The x-axis: tick interval = 1 s, 5 pixels/tick. The y-axis: unit = packets/tick, scale = 10.***

4.2.3. BCNET Traffic Protocol Hierarchy

BCNET Traffic Protocol Hierarchy statistics of collected traffic are shown in Table 2. Each entry (row) consists of the protocol's name, the percentage of protocol packets relative to total number of packets captured, the number of packets, and the number of bytes. From 511,820 packets, 260,639 (50.9%) are BGP packets, 257,285 (50.3%) are TCP ACK packets, and 6,104 (1.2%) are piggyback ACKs. Packets originate from multiple protocols. Therefore, a packet may be attributed to more than one protocol. Protocol layers may consist of packets that do not contain any higher layer protocol, and therefore, the sum of all higher layer packets may not add to the protocols packet count. A single packet may be counted more than once if it is encapsulated by multiple protocols.

Table 2. ***Protocol Hierarchy of the Packets Captured.***

Protocol	Packets%	Packets	Bytes
Ethernet/IP/TCP	100	511,820	98,292,937
BGP	50.92	260,639	79,628,747

4.2.4. BCNET Network Endpoints

BCNET network endpoints are the source and destination addresses of a specified protocol layer. There were six BCNET transit exchanges (BGP peers) captured in the data collection. The network endpoints for data collected in 48 hours between December 20, 2010 and December 22, 2010 were 72.51.24.189, 72.51.24.190, 64.251.87.209, 64.251.87.210, 206.108.83.66, and 206.108.83.70. For each IP address of a BGP peer, there were various TCP connection statistics such as the port number and the number of packets transmitted and received, as shown in Table 3. The data is exchanged between two addresses at a particular time and the number of bytes transmitted at the source address is the same as the number of bytes received at the destination address.

Table 3. Statistics of the Captured TCP Endpoints.

Address	Port	Packets	Transmitted Bytes	Received Bytes
72.51.24.189	bgp	401,721	55,894,998	14,941,356
72.51.24.190	58268	401,721	14,941,356	55,894,998
64.251.87.209	bgp	70,069	12,426,684	2,569,605
64.251.87.210	62844	70,069	2,569,605	12,426,684
206.108.83.66	bgp	40,030	1,500,045	10,960,249
206.108.83.70	51899	40,030	10,960,249	1,500,045

4.2.5. BCNET Traffic Service Response Time

The traffic service response time is defined as the time between a request and the corresponding response. The flow graph of the captured BGP peers traffic is shown in Figure 12. It includes the source address, destination address, TCP port number, TCP message (ACK), and type of the BGP message (*open, update, notification* or *keepalive)*.

The flow graph shows the six BGP peers participating in the traffic exchange. At time 0.000 s, an *update* message is sent from source address 72.51.24.189 to the destination address 72.51.24.190 and the destination sends an acknowledgement back to the source implying that it is ready for traffic exchange. An exchange of data happens between only two BGP peers at a time. When the addresses 64.251.87.209 and 64.251.87.210 exchange data among themselves, there is no data exchanged between 72.51.24.189 and 72.51.24.190 or between 206.108.83.66 and 206.108.83.70. At 5.281 s, the destination address 72.51.24.190 sends a *keepalive* message to source address

72.51.24.189 to confirm that the link between the two is operating. When the destination address receives the acknowledgement, it knows that the link is active and it resumes the data exchange again.

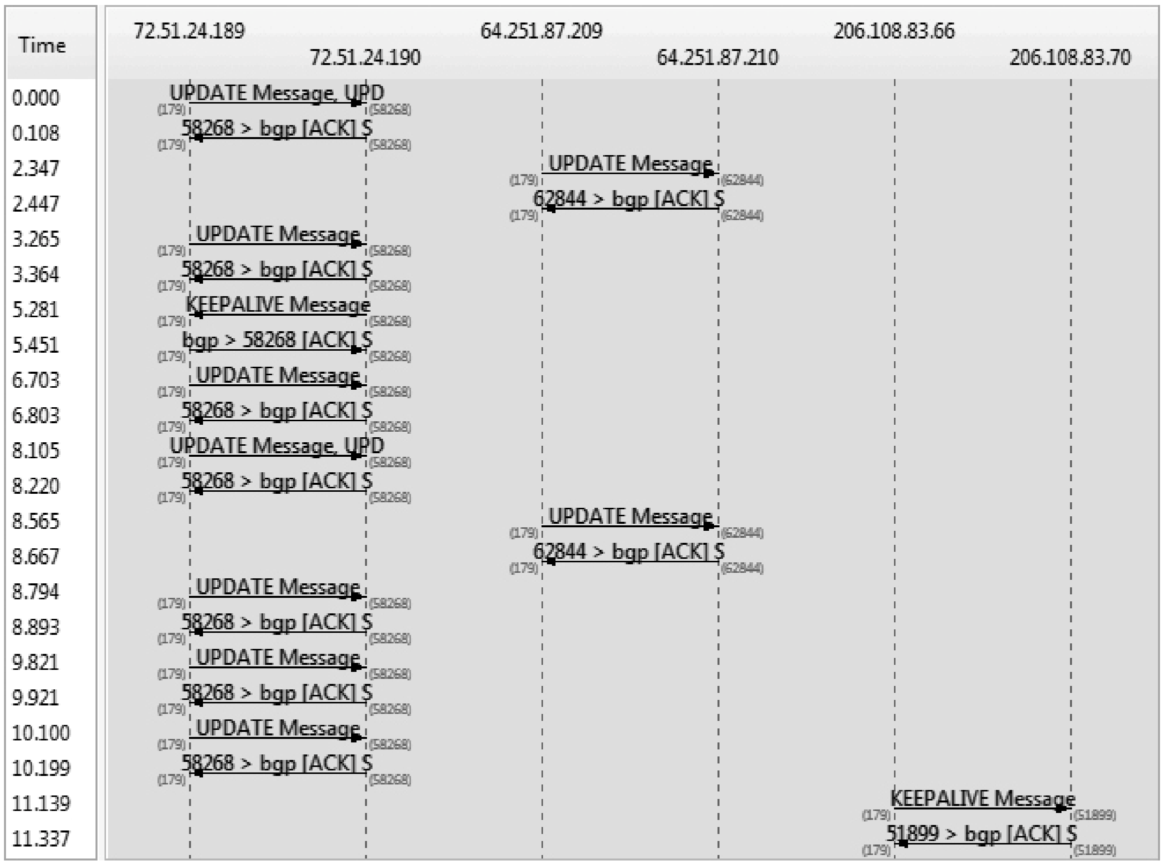
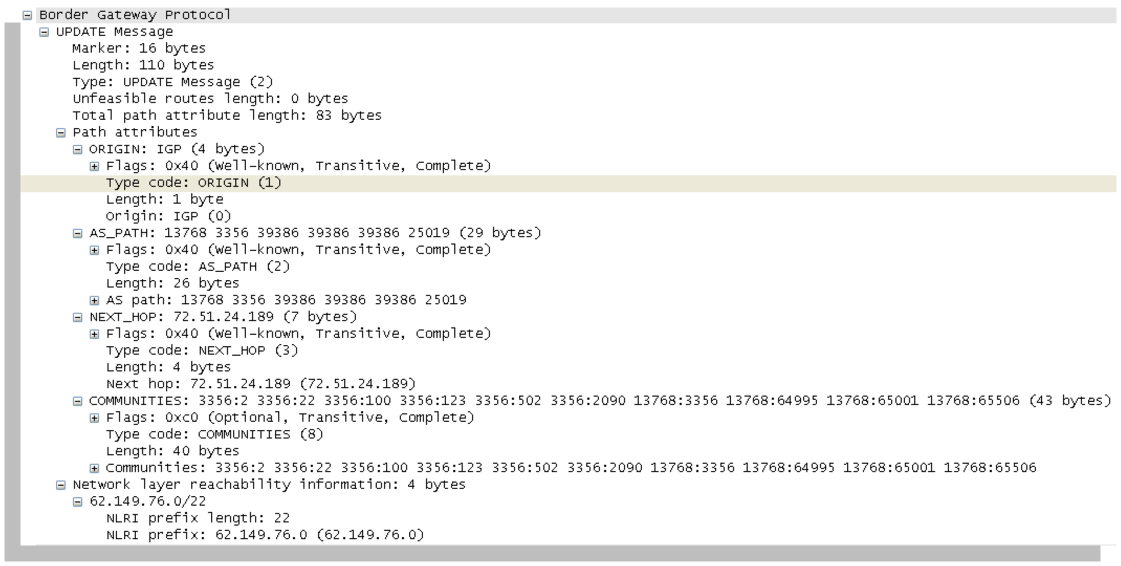

Time
72.51.24.189
72.51.24.190
64.251.87.209
64.251.87.210
206.108.83.66
206.108.83.70
0.000 UPDATE Message, UPD (179) (58268)
0.108 58268 > bgp [ACK] S (179) (58268)
2.347 UPDATE Message (179) (62844)
2.447 62844 > bgp [ACK] S (179) (62844)
3.265 UPDATE Message (179) (58268)
3.364 58268 > bgp [ACK] S (179) (58268)
5.281 KEEPALIVE Message (179) (58268)
5.451 bgp > 58268 [ACK] S (179) (58268)
6.703 UPDATE Message (179) (58268)
6.803 58268 > bgp [ACK] S (179) (58268)
8.105 UPDATE Message, UPD (179) (58268)
8.220 58268 > bgp [ACK] S (179) (58268)
8.565 UPDATE Message (179) (62844)
8.667 62844 > bgp [ACK] S (179) (62844)
8.794 UPDATE Message (179) (58268)
8.893 58268 > bgp [ACK] S (179) (58268)
9.821 UPDATE Message (179) (58268)
9.921 58268 > bgp [ACK] S (179) (58268)
10.100 UPDATE Message (179) (58268)
10.199 58268 > bgp [ACK] S (179) (58268)
11.139 KEEPALIVE Message (179) (51899)
11.337 51899 > bgp [ACK] S (179) (51899)

Figure 12. Flow Graph of Collected Traffic. Shown are Time Stamps of Correspondence Between BGP Peer Routers.

5. BGP Attributes

A BGP attribute describes the characteristics of a particular prefix. They can be either transitive or non-transitive. Some of the BGP attributes are mandatory, while others are optional. The BGP and its attributes for the data collected from BCNET are shown in Figure 13. The total path attribute length was 83 bytes in this particular packet exchange. Some of the BGP attributes such as origin, AS_path, and next_hop are shown. BGP characteristics of BCNET traffic are captured by applying Wireshark BGP display filters in the InputOutput Graphs for the collected packets. Examples of display filters are: bgp.type, bgp.next_hop, bgp.origin,bgp.local_pref, bgp.community_as, bgp.as_path, and bgp.multi_exit_disc.



```
⊟ Border Gateway Protocol
  ⊟ UPDATE Message
      Marker: 16 bytes
      Length: 110 bytes
      Type: UPDATE Message (2)
      Unfeasible routes length: 0 bytes
      Total path attribute length: 83 bytes
    ⊟ Path attributes
      ⊟ ORIGIN: IGP (4 bytes)
        ⊞ Flags: 0x40 (Well-known, Transitive, Complete)
          Type code: ORIGIN (1)
          Length: 1 byte
          Origin: IGP (0)
      ⊟ AS_PATH: 13768 3356 39386 39386 39386 25019 (29 bytes)
        ⊞ Flags: 0x40 (Well-known, Transitive, Complete)
          Type code: AS_PATH (2)
          Length: 26 bytes
        ⊞ AS path: 13768 3356 39386 39386 39386 25019
      ⊟ NEXT_HOP: 72.51.24.189 (7 bytes)
        ⊞ Flags: 0x40 (Well-known, Transitive, Complete)
          Type code: NEXT_HOP (3)
          Length: 4 bytes
          Next hop: 72.51.24.189 (72.51.24.189)
      ⊟ COMMUNITIES: 3356:2 3356:22 3356:100 3356:123 3356:502 3356:2090 13768:3356 13768:64995 13768:65001 13768:65506 (43 bytes)
        ⊞ Flags: 0xc0 (Optional, Transitive, Complete)
          Type code: COMMUNITIES (8)
          Length: 40 bytes
        ⊞ Communities: 3356:2 3356:22 3356:100 3356:123 3356:502 3356:2090 13768:3356 13768:64995 13768:65001 13768:65506
    ⊟ Network layer reachability information: 4 bytes
      ⊟ 62.149.76.0/22
          NLRI prefix length: 22
          NLRI prefix: 62.149.76.0 (62.149.76.0)
```

Figure 13.   BGP Messages and their Path Attributes.

Clustering analysis and detection of anomalies are two important features that may be used to improve network performance and the performance of the routing protocols. The clustering analysis and anomalies are discussed in Sections 5.12 and 5.13 of this Chapter. Clustering analysis is a technique that identifies objects that have similar properties or similar characteristics. It may be used for data analysis and is considered a form of classification. There are many different types of clustering technique that may be used for clustering objects. Clustering algorithms are broadly classified into two types: hierarchical and non-hierarchical algorithms. Anomaly detection refers to the problem of detecting patterns in data that do not match the expected behavior. Anomalies may found in data due to malicious activity, network breakdown, or system failure.

5.1. AS_Path

The AS_path attribute identifies the autonomous systems through which routing information carried in the *update* message is passed. The AS_path is empty when the first route is inserted in BGP. It describes the complete set of AS numbers passed in order to reach any particular network. The AS_path includes numbers of all the autonomous systems on the path between the source and the destination. It prevents routing loops and applies policy decisions based on the presence of certain ASes. Each AS path segment is represented by a triple tuple, which consists of <path segment type, path segment length, and path segment value>. The AS_path attribute helps BGP to determine the best path to route packets. There were three ASes recognized in the BCNET traffic collection in the time period between December 20, 2010 and December 22, 2010 and those ASes were:

- AS 852 (Telus Advanced Communications)
- AS 6327 (Shaw Cable systems)
- AS 13768 (Peer 1 Networks Inc.).

An AS_path is a sequence of intermediate ASes between source and destination routers that form a directed route for packets to travel. Neighboring ASes use BGP to exchange update messages and to reach different AS prefixes. After each router makes a new local decision based on the best route to a destination, it sends that route along with distance metrics and path attributes to each of its peers. As the information travels through the network, each router along the path prepends its unique AS number to a list of ASes in the BGP message [35]. The network traffic for AS 852 and AS 13768 for a time period of 48 hours is shown in Figure 14 and Figure 15, respectively.

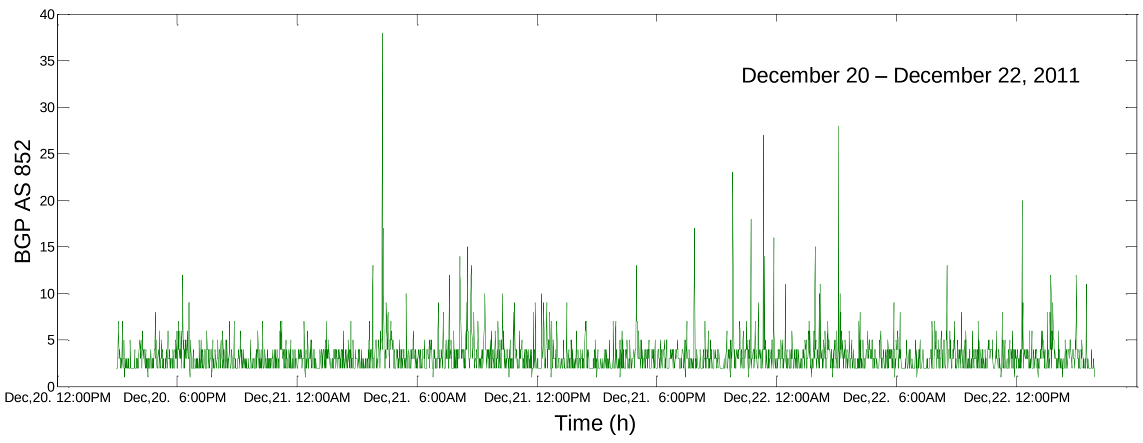


Figure 14. Network Traffic: AS 852.

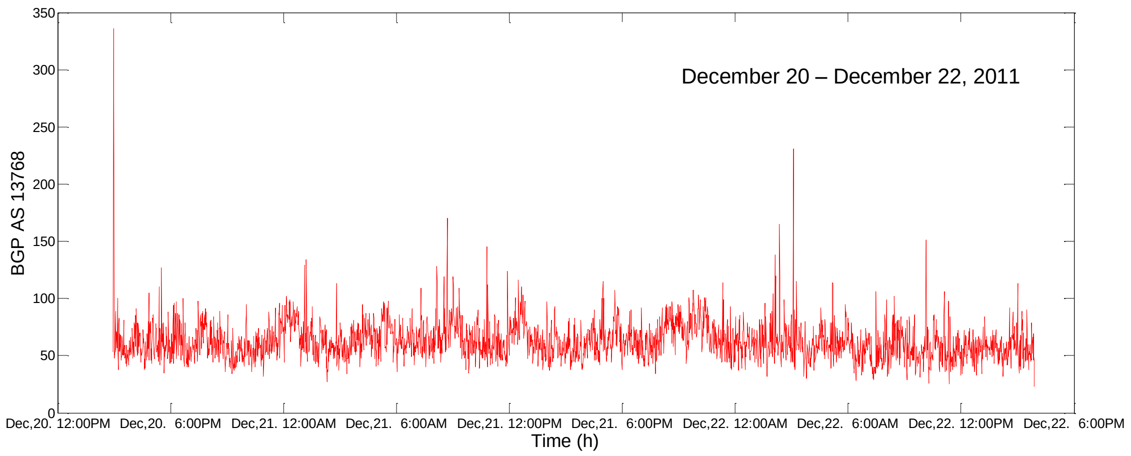


Figure 15. Network Traffic: AS 13768.

The mean of dataset is calculated as the sum of all the observed outcomes of the sample divided by the total number of events. The mean of AS 13768 is 62.89. Median is the middle score, defined as the average of the two middles if the number of outcomes is even. It is 61 for AS 13768. The number with the highest frequency in an event is called a mode. The mode of AS 13768 is 57 in the data collected between December 20 and December 22, 2010. The standard deviation illustrates how close the entire set of data is to the average value. It is 17.03 for AS 13768. The range is 313. The total number of packets is 512,672. The minimum and maximum values are 23 and 336, respectively. The AS 13768 has 588 connections with other ASes, as shown in Figure 16. The network traffic for AS 852 is represented in Figure 16. It has 155 connections with other ASes. The total number of packets is 511,820. The minimum and maximum values are 79 and 645, respectively. The mean, median, and mode are 177.10, 172, and 162, respectively. The standard deviation and range are 35.69 and 566, respectively.

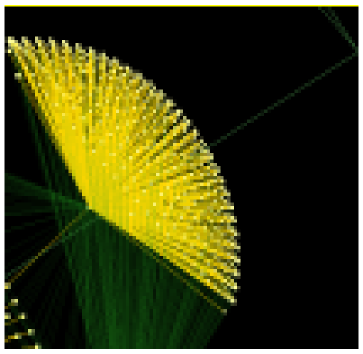


Figure 16. Number of Connections AS 13678 has with other ASes.

The network traffic for AS 6327 is shown in Figure 17. The total number of packets is 30,653, and the minimum and the maximum values are 4 and 96, respectively. The mean and mode are 10.61 and 10, respectively.

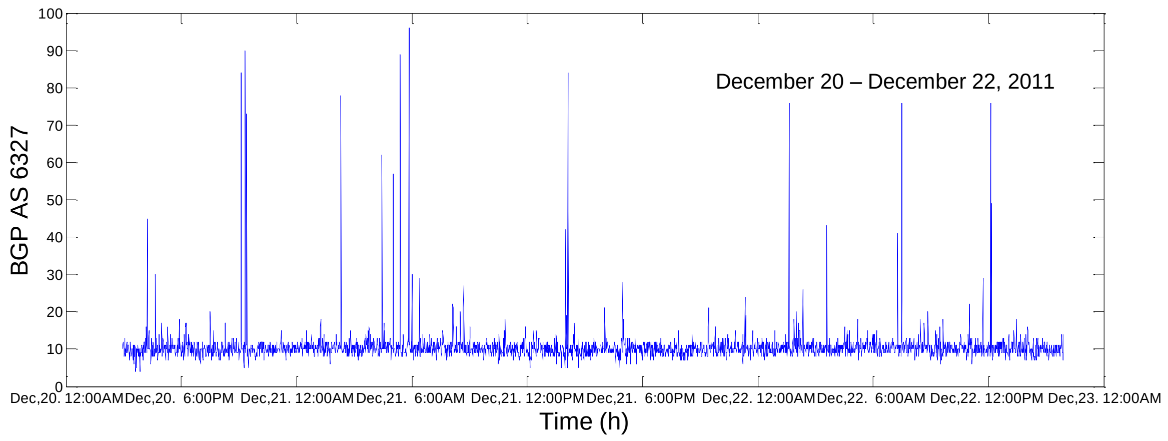


Figure 17. Network Traffic: AS 6327.

Standard deviation is 17.03 and the range is 313. AS 6327 (Shaw cable systems) has 683 connections with other ASes, as shown in Figure 18. The mean, median, standard deviation, and mode for AS 852, AS 6327, and AS 13678 were as expected. We did not encounter any unexpected behavior.

Figure 18. Number of Connections for AS 6327 with other ASes.

5.2. Message Attribute

There are four types of BGP message attributes: *open*, *update*, *keepalive*, and *notification*. 88% of the total messages were *update* messages and the remaining were *keepalive* messages. Therefore, we discuss here *update* and *keepalive* messages only.

Update messages transfer routing information between BGP peers. The information carried by the update packet may be used to construct a graph describing the relationships between various ASes. An *update* message advertises a single feasible route to a peer or withdraws multiple impossible routes from service. An *update* message may simultaneously advertise a realistic route and withdraw multiple routes from service. A total of 230,424 BGP *update* messages were identified between December 20, 2010 and December 22, 2010. The statistics are shown in Table 4.

Table 4. Statistics for Update Messages.

Minimum	0 bits
Maximum	118 bits
Mean	11.7412 bits
Median	11 bits
Mode	10 bits
Standard deviation	7.1972 bits
Range	118 bits

A *keepalive* message is sent by one router to another to confirm that the link between the two is operating and to prevent the link from being broken. If no reply is received after a signal is sent, the link is assumed to be broken and future data is routed via another path until the link is again available. The maximum time between two *keepalive* messages should be one third of the hold time interval and they should not be sent more frequently than one per second. When the BGP peer waits for the next *update* message, it remains idle. Each BGP device maintains a hold timer to keep a track of how long it has been on hold. The length of the hold timer is part of session setup that is established using *open* messages. To ensure that the timer does not expire when no *update messages* are sent for a long time, each peer periodically sends a BGP *keepalive* message. Statistics for 30,462 *keepalive* messages are shown in Table 5. As expected, the *keepalive* messages comprise less than 12.5% of the total BGP *update* messages.

Table 5. Statistics for Keepalive Messages.

Minimum	0 bits
Maximum	32 bits
Mean	0.2875 bits
Median	0 bits
Mode	0 bits
Standard deviation	1.4025 bits
Range	32 bits

5.3. Origin Attribute

The origin is a mandatory attribute that defines the origin of the path information. Its value should not be changed by any other BGP speaker. The origin attribute may assume one of three values: Interior Gateway Protocol (IGP), Exterior Gateway Protocol, and Incomplete.

Interior Gateway Protocol (IGP) is a routing protocol used to exchange routing information within an autonomous system. IGP is indicated by an “i” in the BGP table. The network traffic for 210,414 IGP packets is shown in Figure 19.

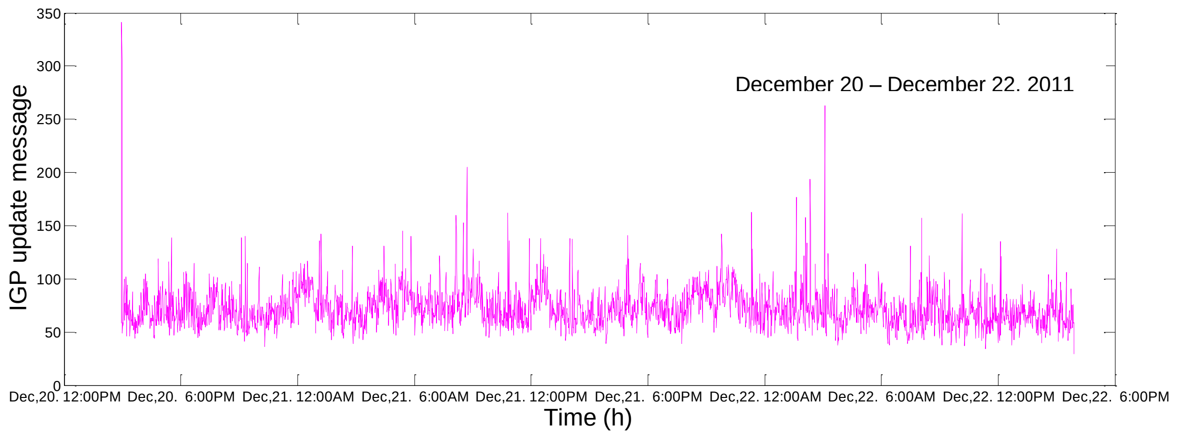


Figure 19. Network Traffic: 210,414 IGP Packets were collected from December 20 to December 22, 2010.

The statistics for IGP packets in the time period between December 20, 2010 and December 22, 2010 are shown in Table 6. The IGP, EGP, and Incomplete account for 85.82%, 0.003%, and 13.84%, of the total number of *update* messages, respectively.

Table 6. Statistics for IGP Packets.

Minimum	29 bits
Maximum	341 bits
Mean	72.8076 bits
Median	70 bits
Mode	66 bits
Standard Deviation	18.9766 bits
Range	312 bits

Exterior Gateway Protocol (EGP) is a routing protocol used to transport information to other BGP-enabled systems in different autonomous systems. EGP is indicated by an “e” in the BGP table. The total number of EGP packets is 822. The network traffic is shown in Figure 20. The EGP became obsolete when the Internet migrated from EGP to BGP [36]. The statistics for the EGP packets in the BGP traffic collected are shown in Table 7.

Table 7. Statistics for EGP Packets.

Minimum	0 bits
Maximum	32 bits
Mean	0.2875 bits
Median	0 bits
Mode	0 bits
Standard Deviation	1.4025 bits
Range	32 bits

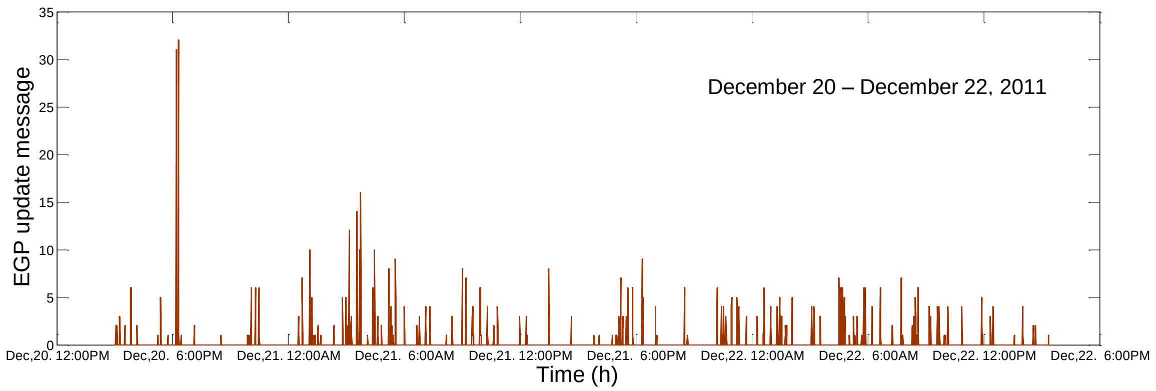


Figure 20. Network Traffic: 822 EGP Packets were recognised in the time period between December 20 and December 22, 2010.

When the network layer reachability information is learned by unfamiliar means and the route is unknown the message is known as *incomplete* message. Incomplete is indicated with “?” in the BGP table. Table 8 shows the statistics for incomplete packets. There were 33,932 incomplete packets in this case and the network traffic is shown in Figure 21.

Table 8. Statistics for Incomplete Packets.

Minimum	0 bits
Maximum	118 bits
Mean	11.7412 bits
Median	11 bits
Mode	10 bits
Standard Deviation	7.1972 bits
Range	118 bits

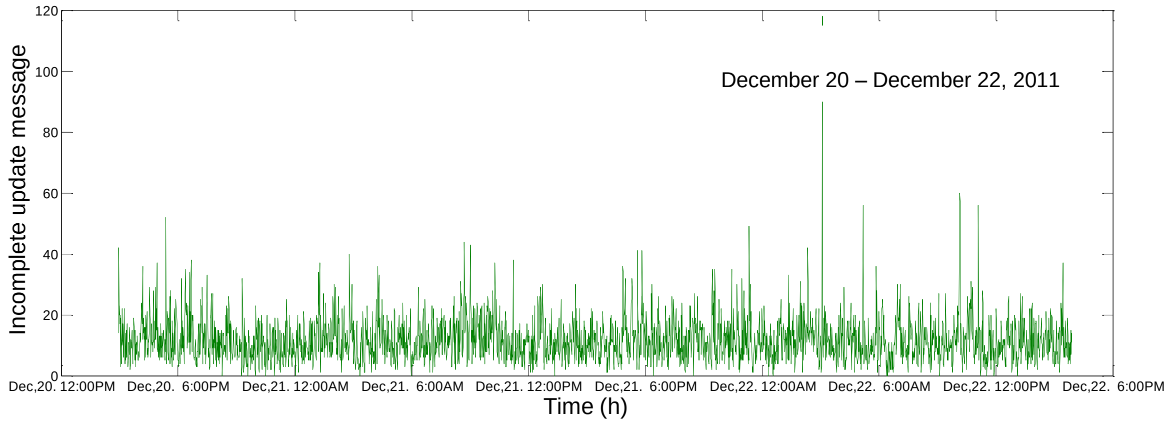


Figure 21. Network Traffic: 33,932 Incomplete Packets.

There were a total of 245,168 origin packets in the data collected over a period of 48 hours from December 20 to December 22, 2010. Out of the total number of origin packets, 210,414 were IGP packets, 33,932 were incomplete packets, and 822 were EGP packets, as shown in Figure 23.

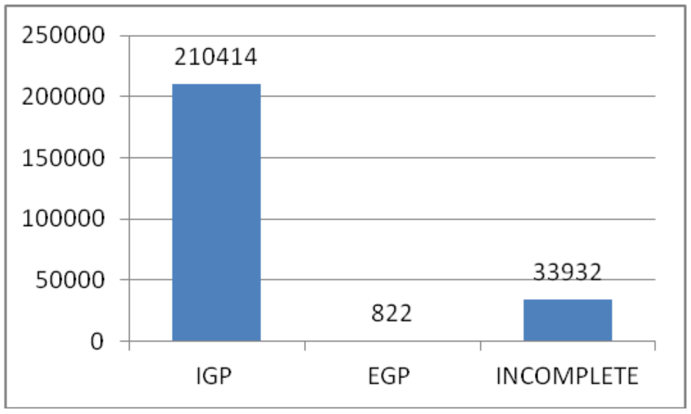


Figure 22. Distribution of BGP Origin Attributes.

An example of a packet where origin attribute assumes value IGP and incomplete values is shown in Figure 22. In this example, Router A reaches the address 170.10.20.1 through 300 i, where "300 i" means the next AS path is 300 and the origin of the route is IGP. Router B also reaches the address 190.10.50.1 through i. This "i" means that the entry is in the same AS and the origin is IGP [36]. Router C reaches 150.10.30.1 through 100 i and "100 i" means that the next AS is 100 and the origin is

IGP. Router C also reaches 190.10.0.0 through 100 ? and the "100 ?" means that the next AS is 100 and that the origin is incomplete and comes from a fixed route.

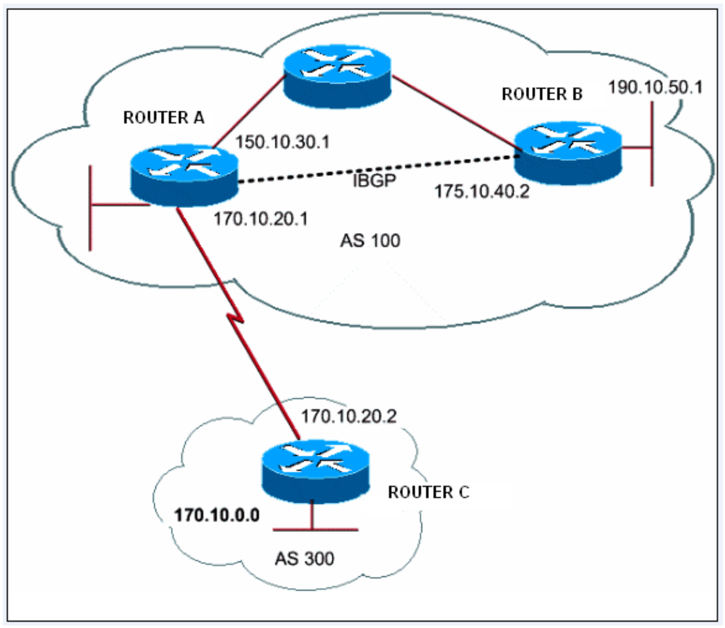


Figure 23. An Example of Incomplete and IGP Origin Attribute [36].

The origin attribute is set when the route is first introduced to the BGP. If information about an IP subnet is inserted using network command or via aggregation, the origin attribute is set to IGP. IGP is a protocol for exchanging routing information between hosts with routers within an AS. Figure 24 shows the distribution of the origin path attributes (IGP, EGP, and incomplete).

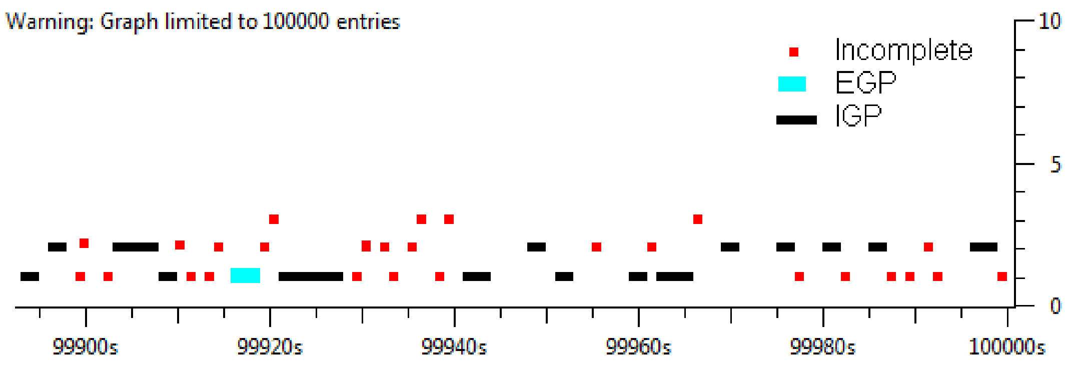


Figure 24. The Graph Shows the Distribution of the Origin Attribute (IGP, EGP, and INCOMPLETE).

5.4. Next_Hop

The next_hop is another mandatory attribute that defines the router IP address that should be used as the next hop to the destinations provided in the *update* message. Generally, this attribute is chosen so that the shortest available path is taken. The immediate next-hop address is determined by performing a recursive route lookup operation for the IP address in the next_hop attribute. Recursive route lookup is the second route lookup that is required in order to determine the exit path for traffic directed towards the destination.

5.5. Multiple_Exit_Discriminator

The multiple_exit_discriminator is an optional attribute that is used on external links to distinguish among multiple exit or entry points to the same neighboring autonomous systems. A BGP speaker should implement a mechanism that allows the attribute to be removed from a route. The absence of MED attribute implies that MED value is zero.

5.6. Local_Preference

The local_preference is a local attribute used in the route selection process. It is included in all *update* messages that a BGP speaker sends to other internal neighbors. A BGP speaker calculates the degree of preference for each external route based on the locally-configured policy. It includes the degree of preference when advertising a route to its internal peers. It defines the preferred route when multiple routes to the same destination are available.

5.7. Atomic_Aggregate

When a BGP speaker aggregates several routes for the purpose of advertisement to a particular peer, the AS_path of the aggregate route usually includes an AS_set compiled from the set of ASes from which the aggregate was formed.

5.8. Aggregator

Aggregator is an optional transitive attribute that may be included in updates that are formed by aggregation. It does not influence path selection and is used for debugging purposes. This attribute contains the last AS number that formed the aggregate route, followed by the IP address of the BGP speaker that formed this route.

5.9. TCP Round Trip Time

Round-trip time (RTT) is the length of the time that it takes for a signal to be sent for an acknowledgment of that signal to be received. When a host transmits a TCP packet to its peer, it should wait a certain time for an acknowledgment. If the reply does not arrive within the expected period, the packet is assumed to have been lost and the data are retransmitted. The time sufficient for a reply over an Ethernet cable should be no more than a few microseconds. If the traffic flows over the wide-area Internet, a second or two seconds are reasonable during peak utilization times. Network traffic for Transmission Control Protocol RTT is shown in Figure 25.

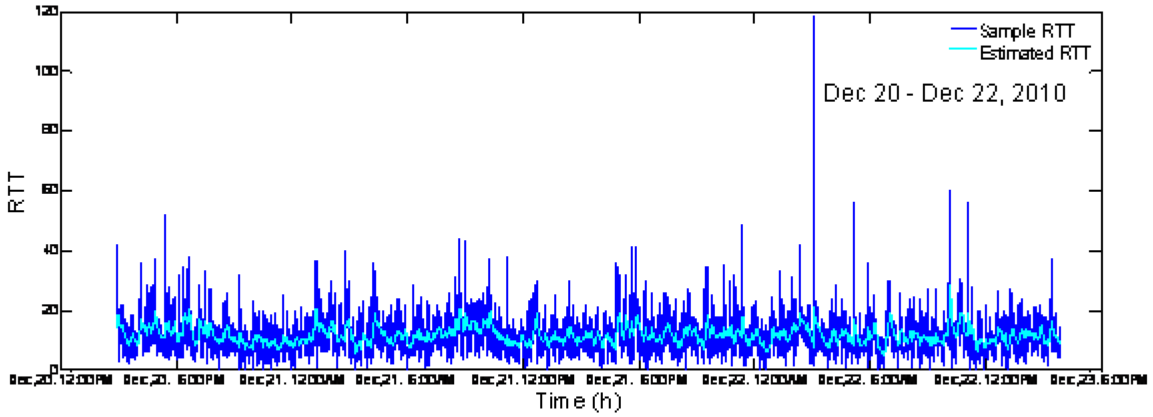


Figure 25. Network Traffic: Transmission Control Protocol RTT.

RTT estimation is one of the most important performance parameters in a TCP exchange, especially in the case of a large file transfer. All TCP implementations eventually drop packets and retransmit them, no matter how good the quality of the link. If the RTT estimate is too low, packets are retransmitted unnecessarily; if it is too high, the connection may sit idle while the host waits for a timeout.

The average RTT is approximately 11.7 ms. The RTT standard deviation is 7.19 ms and 2.75 ms for the sample and estimated RTT, respectively. Table 9 shows the statistics for sample RTT while Table 10 shows statistics for estimated RTT.

Estimated RTT is a weighted average of the sample RTT values. It is calculated as [37]:

$$\text{EstimatedRTT} = (1- \alpha) \cdot \text{EstimatedRTT} + \alpha \cdot \text{SampleRTT}.$$

The recommended value of α is 0.125.

Therefore, EstimatedRTT = 0.875 · EstimatedRTT + 0.125 · SampleRTT.

Table 9. Statistics for Sample TCP RTT.

Minimum (s)	0
Maximum (s)	118
Mean (s)	11.7412
Median (s)	11
Mode (s)	10
Standard Deviation (s)	7.1972
Range (s)	118

Table 10. Statistics for Estimated TCP RTT.

Minimum (s)	4.9433
Maximum (s)	28.6384
Mean (s)	11.7506
Median (s)	11.3748
Mode (s)	4.9433
Standard Deviation (s)	2.7528
Range (s)	23.6951

5.10. TCP Throughput

TCP considers two most important factors: TCP window size and the round trip latency to transfer data. If the TCP window size and the round trip latency are known, the maximum possible throughput of a data transfer between two hosts may be calculated regardless of the bandwidth using the expression:

TCP-Window-Size-in-bits / Latency-in-seconds = Bits-per-second throughput.

Instantaneous throughput is the rate (bps) at which a host receives the packets. If the packets consist of F bits and the transfer takes T seconds for the host to receive all F bits, then the average throughput of the packets transfer is F/T bps. The TCP throughput graph shown in Figure 26 illustrates that the average throughput of the collected data is 177.1 packets/min and that the maximum throughput is 645 packets/min. Various factors such as link speed, propagation delay, window size, link reliability, or congestion of network and intermediate devices affect the throughput of TCP.

TCP is a connection-oriented service that guarantees reliable, in-order delivery of data. TCP has two mechanisms: flow control mechanism and congestion-control mechanism. The flow-control mechanism ensures that a sender does not overrun the buffer at the receiver, while its congestion-control mechanism tries to prevent too much data from being injected into the network. While the size of the flow-control window is static, the size of the congestion window evolves over time according to the status of the network [38].

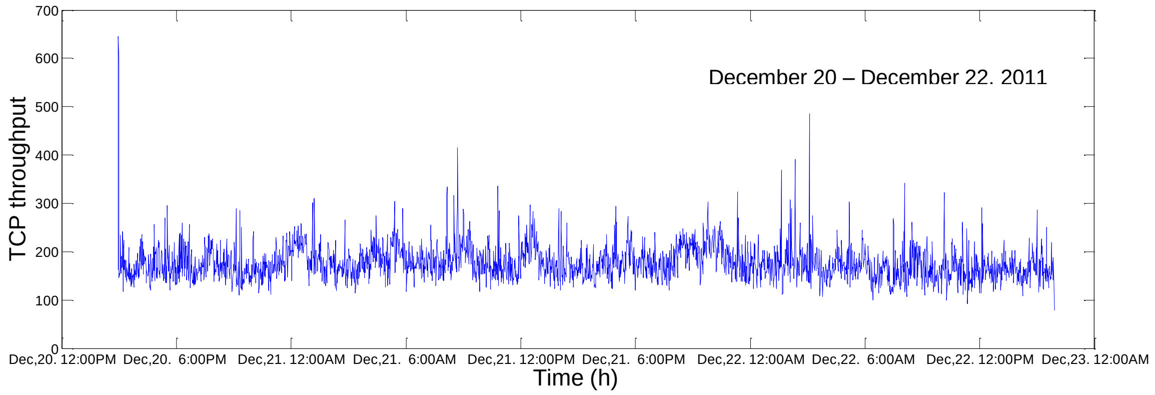


Figure 26. ***TCP Throughput of the BCNET Traffic Collected from December 20 to December 22, 2010 had an Average of 177.1 packet/min.***

The TCP congestion mechanism controls its packet transmission rate by changing the window size in response to network congestion. A TCP sender additively increases its rate when it identifies that the end-to-end path is congestion-free. It multiplicatively decreases its rate when it detects that the path is congested. TCP congestion control is also referred as additive-increase, multiplicative-decrease (AIMD) algorithm because of its additive and multiplicative nature [39].

The four phases of congestion control algorithms are *slow start*, *congestion avoidance*, *fast retransmit*, and *fast recovery*, as shown in Figure 27. Congestion control algorithms use two TCP variables, the congestion window size (*cwnd*) and the receiver's advertised window (*rwnd*). These variables control the amount of data that may be transmitted in the network.

The slow start threshold (*ssthresh*) determines when to use the slow start or congestion avoidance algorithm. After the three-way handshake is complete, the *cwnd* is equal to initial window (IW).

$$\text{IW} = \min(4 \times \text{SMSS}, \max(2 \times \text{SMSS}, 4380 \text{ bytes})).$$

where SMSS is sender maximum segment size.

A TCP sender triggers fast retransmit and fast recovery algorithms when it detects three duplicate ACKs that indicate congestion. In fast retransmit, a TCP sender retransmits data without waiting for the retransmission timeout (RTO) timer to expire and *ssthresh* is assigned a new value as:

$$\text{ssthresh} = \text{cwnd}/2.$$

In fast recovery, a TCP sender adjusts the *cwnd* for all segments buffered by a TCP receiver:

$$\text{cwnd} = \text{ssthresh} + 3 \times \text{SMSS}.$$

During RTO period, the *ssthresh* value is set to:

$$\text{ssthresh} = \max(\text{flightsize} / 2, 2 \times \text{SMSS}).$$

where *flightsize* is the size of outstanding data in the network.

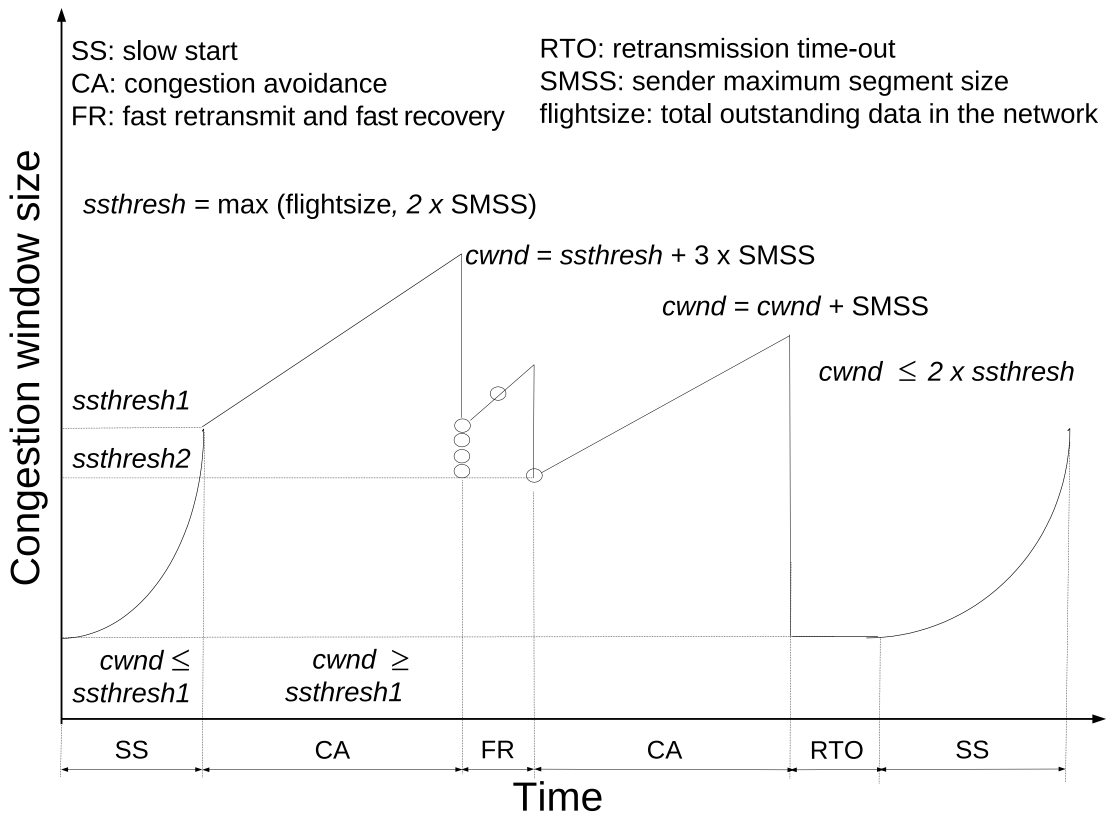

SS: slow start
CA: congestion avoidance
FR: fast retransmit and fast recovery
RTO: retransmission time-out
SMSS: sender maximum segment size
flightsize: total outstanding data in the network
Congestion window size
ssthresh = max (flightsize, 2 x SMSS)
cwnd = ssthresh + 3 x SMSS
cwnd = cwnd + SMSS
cwnd ≤ 2 x ssthresh
ssthresh1
ssthresh2
cwnd ≤ ssthresh1
cwnd ≥ ssthresh1
SS
CA
FR
CA
RTO
SS
Time

Figure 27. ***TCP Congestion Control Algorithms. The Congestion Window Size is Determined by the Congestion Control Algorithm and the Mechanism Used to Indicate Congestion.***

5.11. TCP Window Size

The amount of data that a host may accept without the acknowledgement from the sender is known as the TCP window size. If the sender has not received an acknowledgement for the first packet it sent, it will stop and wait; if this wait exceeds a certain time limit, it may even retransmit. This is how TCP achieves reliable data transmission. TCP transmits data up to the window size before waiting for the acknowledgements. However, the full bandwidth of the network may not always get used. The TCP window size for 200 samples of data from December 20, 2010 to December 22, 2010 is shown in Figure 28.

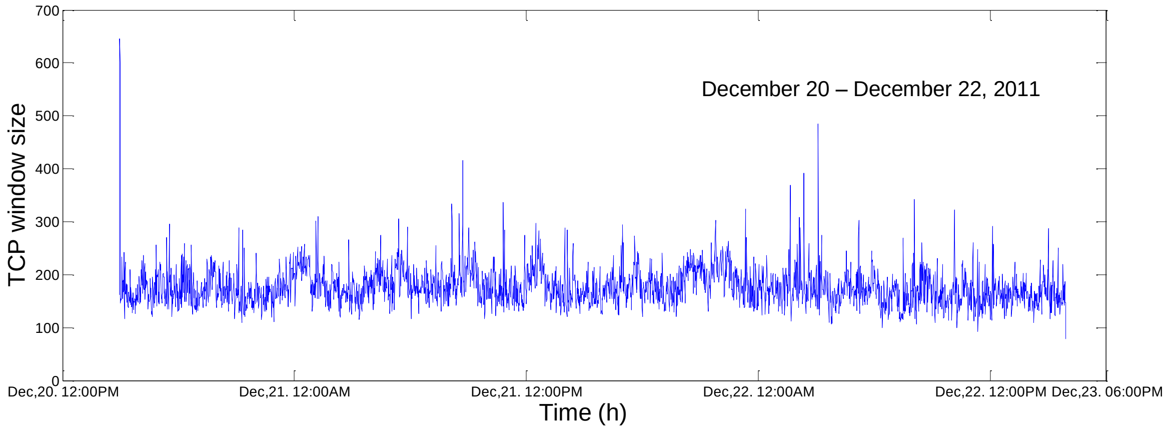


Figure 28. TCP Window Size of the BCNET Traffic for 200 Samples.

5.12. Anomalies

The BGP *update* message may contain anomalies. It is important to detect and eliminate these anomalies. They may arise both as a result of errors by network operators or malicious attacks. Such incidents include routing loops, policy violations, and incorrect export of routes between neighboring ASes, origin violations and address space hijacks, false announcements claiming non-existent connectivity, or private AS announcements. Anomalies may be either path anomalies or announcement anomalies. The unexpected events that occur in AS path attribute are called path anomalies, while the anomalies that occur in update or withdrawal message are called announcement anomalies. There are various methods to detect anomalies. Any suspicious activity may be categorized as an anomaly.

The selected attributes in the data collected on December 20, 2011 were categorized into volume attributes and AS path attributes. 65% of the selected attributes were volume attributes. Hence, they were more relevant to the anomaly class than the AS-path attributes, which confirmed the enormous effect of BGP anomalies on the volume of the BGP announcements. The prolonged spikes in the number of BGP *update* messages are due to the routing instability and affect the inter-domain routing. Self-similarity and long-range dependence have been observed and estimated in various types of network data traffic such as LAN, WAN, and WWW. BGP routing *updates* also exhibit self-similarity when compared to traditional data traffic. Forwarding instable routes may cause packet losses and delays in the routing convergence. Hence, detecting anomalies is an important aspect of BGP *update* messages.

Detecting anomalous BGP-route advertisements is essential for improving the security and robustness of the Internet's inter-domain-routing system [40]. Anomalies such as Slammer [41], Nimda [42], and Code Red I [43] affect performance of the Internet Border Gateway Protocol (BGP). Statistical and machine-learning techniques may be used to classify and detect BGP anomalies [44], [45]. Detection of anomalies may further be used for classification.

5.13. Clusters

In the telecommunication industry, clustering techniques may be used to identify traffic patterns, detect fraudulent activities, and discover users' mobility patterns. Clustering analysis is used to determine hidden patterns and relationships in data sets. Clustering is defined as the task of assigning a set of objects to groups called clusters so that the objects in the same cluster are more similar (in one way or the other) to each other than to objects in the other clusters. The different types of clustering techniques are hierarchical clustering, k-means clustering, and DBSCAN.

We recognized three clusters in the data collected between December 20, 2010 and December 22, 2010. The three clusters of ASes correspond to the three BCNET transit service providers Telus Advanced Communications (AS 852), Shaw Communications (AS 6327), and Peer 1 Network Inc. (AS 13768). The Cooperative Association for Internet Data Analysis (CAIDA) [46] searches hands-on and educational features of the Internet to examine the Internet infrastructure, performance, usage, and its growth. It encourages an environment in which data can be obtained, analyzed, and shared to improve the Internet. The figure is generated using CAIDA [47] tools. The Walrus 3D hyperbolic display [48] of the BCNET AS topology is shown in Figure 29. Clusters consist of 683, 588, and 155 AS nodes, respectively, as shown in the figure.

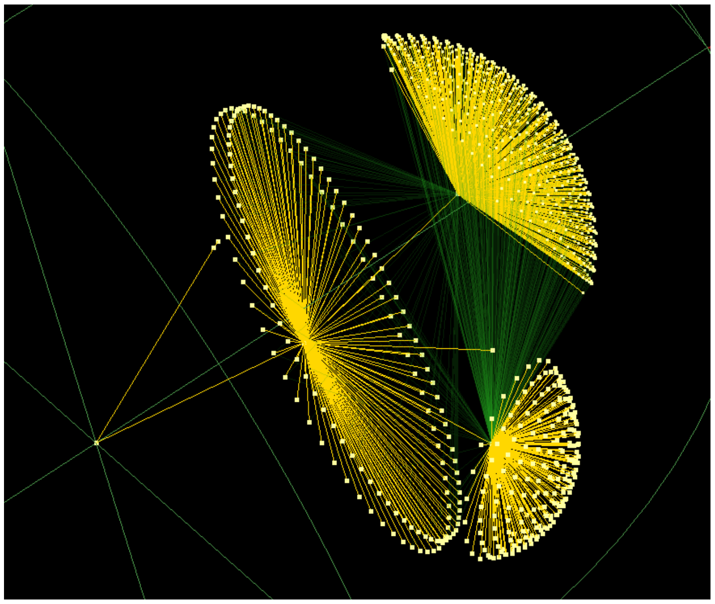


Figure 29. Walrus AS Topology Graph of the Collected BCNET Traffic. The Clusters Correspond to AS 852 (Telus), AS 6327 (Shaw), and AS 13678 (Peer 1 Networks).

The graph consists of 982 nodes, 981 tree-links, and 441 non tree-links. It is created using the value of the BGP AS path attribute in BGP update messages. The AS path attribute is generated by the Best Path Selection algorithm and contains a list of ASes. The graph links reflect a policy relationship between BCNET transit providers and do not necessarily indicate the actual data traffic flow.

6. BGP Update Attributes

We extracted from the BGP *update* messages several attributes described in this section. We used a C# code (in Appendix A and B) to preprocess the readable Multi-threaded Routing Toolkit (MRT) files. Internet Engineering Task Force (IETF) designed the MRT file format to export routing protocol messages, state changes, and routing information base contents. We extracted numerical attributes from BGP traffic and used MATLAB (Appendix C) to generate various graphs.

The C# code performed the basic function of parsing the attributes from the BGP update messages. It separated the update messages received from various peers into different datasets and then parsed these datasets to obtain the attributes. The attributes were computed for one-minute intervals within 24-hour time period.

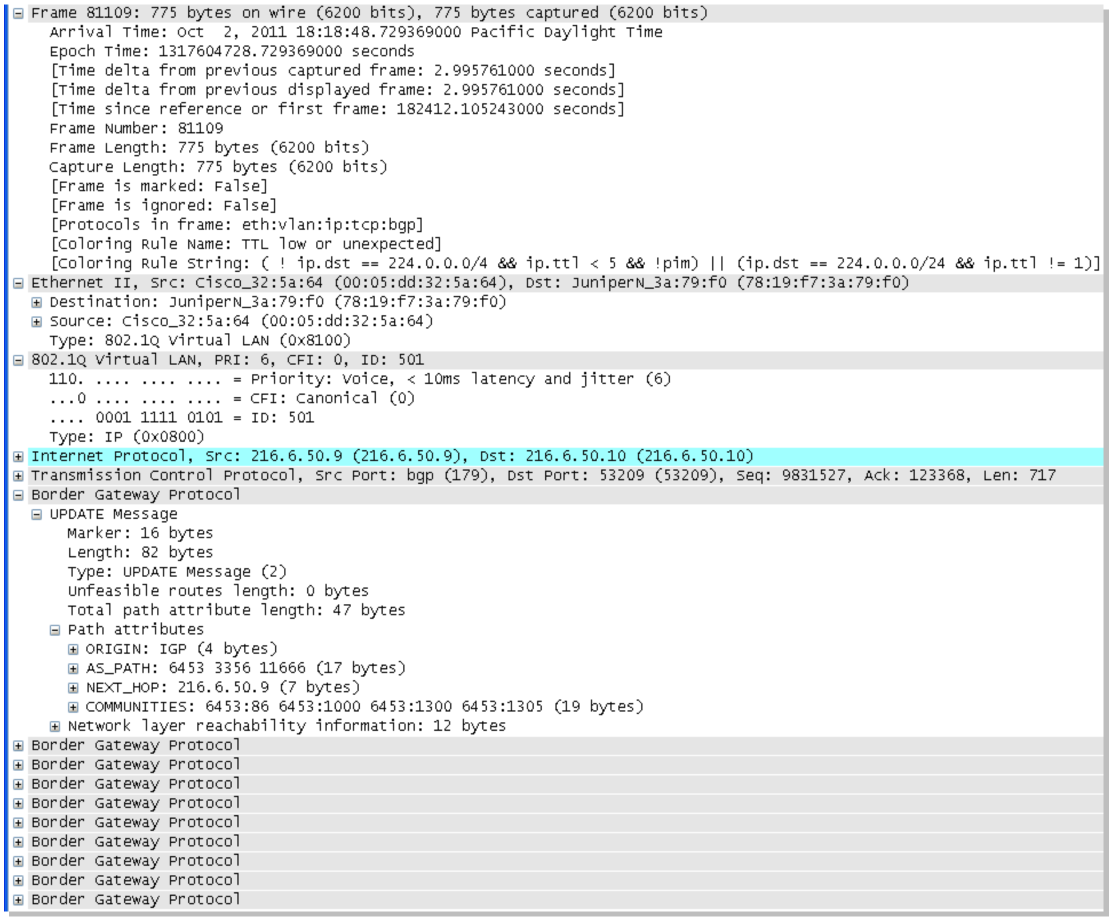
We chose data collected on October 2, November 2, and December 2, 2011 and compared different attributes to emphasize data extraction and data collection. The number of announcements and withdrawals exchanged by neighboring peers were an important feature that occurred during instability periods. The attributes were categorized as volume (number of BGP announcements) and AS-path (maximum edit distance) attributes. The extracted BGP update message attributes are shown in Table 11.The extracted attributes are categorized into two: volume (number of BGP announcements) and AS-path (maximum edit distance) attributes. There were 37 attributes extracted. However, we considered only first 14 attributes. Since the attributes 14 to 33 were calculated using the most frequent values for the maximum edit distance and the maximum AS path length. These values may be used in detecting worms. A number of EGP packets may be present in the case of worms in the traffic traces. Since there were no worms detected on October 2, 2011, November 2, 2011, and December 2, 2011, there were no EGP packets detected.

Table 11. Extracted BGP Update Message Attributes.

Attribute	Definition	Category
1	Number of announcements	volume
2	Number of withdrawals	volume
3	Number of announced NLRI prefixes	volume
4	Number of withdrawn NLRI prefixes	volume
5	Average AS path length	AS-path
6	Maximum AS path length	AS-path
7	Average unique AS path length	AS-path
8	Number of duplicate announcements	volume
9	Number of duplicate withdrawals	Volume
10	Number of implicit withdrawals	Volume
11	Average edit distance	AS-path
12	Maximum edit distance	AS-path
13	Inter-arrival time	volume
14-24	Maximum edit distance=n (7,...17)	AS-path
24-33	Maximum AS path length=n (7,...16)	AS-path
34	Number of IGP packets	volume
35	Number of EGP packets	volume
36	Number of incomplete packets	volume
37	Packet size	volume

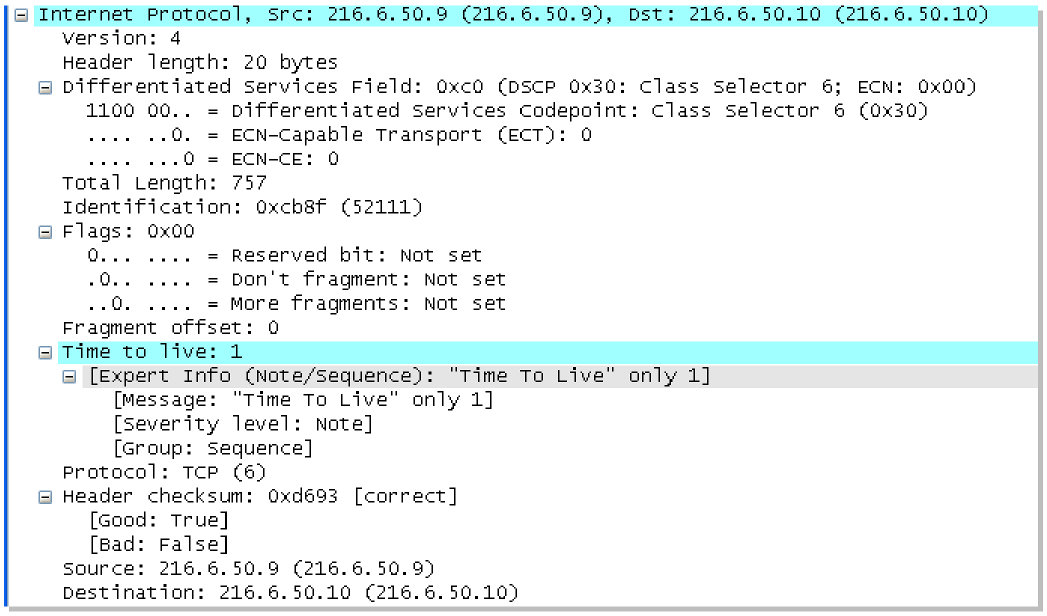
A sample of data captured on October 2, 2011 using Wireshark is shown in Figure 30. Details of various fields are shown, such as the frame number 81109, the frame length 775 bytes, and its arrival time on October 2, 2011 18:18:48.729369000 Pacific Daylight Time. The protocols shown in this particular frame are Ethernet, 802.1Q Virtual Local Area Network (VLAN), IP, TCP, and BGP. The source and destination for Ethernet protocol are Cisco and Juniper routers, respectively. The IP source address is 216.6.50.9 and IP destination address is 216.6.50.10. The source port and destination port of TCP are BGP (I79) and 53209, respectively. In this case, the BGP message is an update message and has a marker of 16 bytes and length of 82 bytes. The path attributes are: Origin is IGP, AS_path is 6453 3356 11666 (17 bytes), Next_hop is 216.6.50.9, and NLRI size is of 12 bytes.

The details for one of the IP packets are shown in Figure 31. The first line shows the source and the destination addresses of the IPv4 packet. *Time to live* (TTL) is an 8-bit field. In the IPv4 header, TTL is the 9th octet of 20 and in the IPv6 header, it is the 8th octet of 40. The maximum value of a TTL field may be 255, which is the maximum value of a single octet. In this particular case, TTL=1.



```
⊟ Frame 81109: 775 bytes on wire (6200 bits), 775 bytes captured (6200 bits)
    Arrival Time: Oct  2, 2011 18:18:48.729369000 Pacific Daylight Time
    Epoch Time: 1317604728.729369000 seconds
    [Time delta from previous captured frame: 2.995761000 seconds]
    [Time delta from previous displayed frame: 2.995761000 seconds]
    [Time since reference or first frame: 182412.105243000 seconds]
    Frame Number: 81109
    Frame Length: 775 bytes (6200 bits)
    Capture Length: 775 bytes (6200 bits)
    [Frame is marked: False]
    [Frame is ignored: False]
    [Protocols in frame: eth:vlan:ip:tcp:bgp]
    [Coloring Rule Name: TTL low or unexpected]
    [Coloring Rule String: ( ! ip.dst == 224.0.0.0/4 && ip.ttl < 5 && !pim) || (ip.dst == 224.0.0.0/24 && ip.ttl != 1)]
⊟ Ethernet II, Src: Cisco_32:5a:64 (00:05:dd:32:5a:64), Dst: JuniperN_3a:79:f0 (78:19:f7:3a:79:f0)
  ⊞ Destination: JuniperN_3a:79:f0 (78:19:f7:3a:79:f0)
  ⊞ Source: Cisco_32:5a:64 (00:05:dd:32:5a:64)
    Type: 802.1Q Virtual LAN (0x8100)
⊟ 802.1Q Virtual LAN, PRI: 6, CFI: 0, ID: 501
    110. .... .... .... = Priority: Voice, < 10ms latency and jitter (6)
    ...0 .... .... .... = CFI: Canonical (0)
    .... 0001 1111 0101 = ID: 501
    Type: IP (0x0800)
⊞ Internet Protocol, Src: 216.6.50.9 (216.6.50.9), Dst: 216.6.50.10 (216.6.50.10)
⊞ Transmission Control Protocol, Src Port: bgp (179), Dst Port: 53209 (53209), Seq: 9831527, Ack: 123368, Len: 717
⊟ Border Gateway Protocol
  ⊟ UPDATE Message
      Marker: 16 bytes
      Length: 82 bytes
      Type: UPDATE Message (2)
      Unfeasible routes length: 0 bytes
      Total path attribute length: 47 bytes
    ⊟ Path attributes
      ⊞ ORIGIN: IGP (4 bytes)
      ⊞ AS_PATH: 6453 3356 11666 (17 bytes)
      ⊞ NEXT_HOP: 216.6.50.9 (7 bytes)
      ⊞ COMMUNITIES: 6453:86 6453:1000 6453:1300 6453:1305 (19 bytes)
    ⊞ Network layer reachability information: 12 bytes
⊞ Border Gateway Protocol
⊞ Border Gateway Protocol
⊞ Border Gateway Protocol
⊞ Border Gateway Protocol
⊞ Border Gateway Protocol
⊞ Border Gateway Protocol
⊞ Border Gateway Protocol
⊞ Border Gateway Protocol
⊞ Border Gateway Protocol
```

Figure 30. A Sample of Data Collected Using Wireshark.



```
⊟ Internet Protocol, Src: 216.6.50.9 (216.6.50.9), Dst: 216.6.50.10 (216.6.50.10)
    Version: 4
    Header length: 20 bytes
  ⊟ Differentiated Services Field: 0xc0 (DSCP 0x30: Class Selector 6; ECN: 0x00)
      1100 00.. = Differentiated Services Codepoint: Class Selector 6 (0x30)
      .... ..0. = ECN-Capable Transport (ECT): 0
      .... ...0 = ECN-CE: 0
    Total Length: 757
    Identification: 0xcb8f (52111)
  ⊟ Flags: 0x00
      0... .... = Reserved bit: Not set
      .0.. .... = Don't fragment: Not set
      ..0. .... = More fragments: Not set
    Fragment offset: 0
  ⊟ Time to live: 1
    ⊟ [Expert Info (Note/Sequence): "Time To Live" only 1]
        [Message: "Time To Live" only 1]
        [Severity level: Note]
        [Group: Sequence]
    Protocol: TCP (6)
  ⊟ Header checksum: 0xd693 [correct]
      [Good: True]
      [Bad: False]
    Source: 216.6.50.9 (216.6.50.9)
    Destination: 216.6.50.10 (216.6.50.10)
```

Figure 31. Details of an Internet Protocol Packet.

6.1. Number of Announcements

BGP announcements are the number of routes that are available for delivery of data from source to the destination. A set of path attributes should be described for number of BGP announcements. BGP *update* messages are sent to every BGP router for which a session has been established having a BGP announcement or a BGP withdrawal. If, according to the route attributes, the received route is better than the current route, the received route replaces the existing route for the same destination in the FIB. Otherwise, the received route is only added to the RIB. The number of announcements on October 2, 2011 shown in Figure 32 is below 20. The minimum and maximum values were 1 and 16, respectively. The mean of announcements on October 2, 2011 was 4.72 packets.

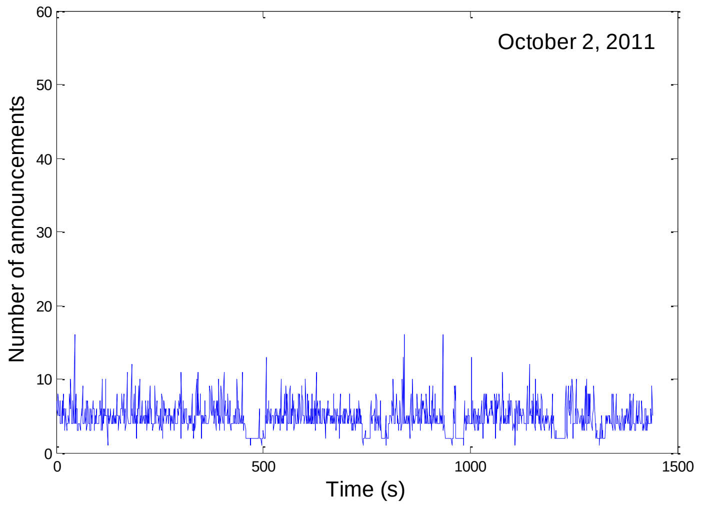


Figure 32. Number of Announcements on October 2, 2011.

The number of announcements on November 2, 2011 had a range of 51 packets as illustrated in Figure 33. The mean of announcements was 3.84 packets over the 24-hour period. The number of announcements had a range of only 15 packets compared to the range of 51 packets on November 2, 2011. However, the mean in the first case is greater than the mean of announcements of November 2, 2011.

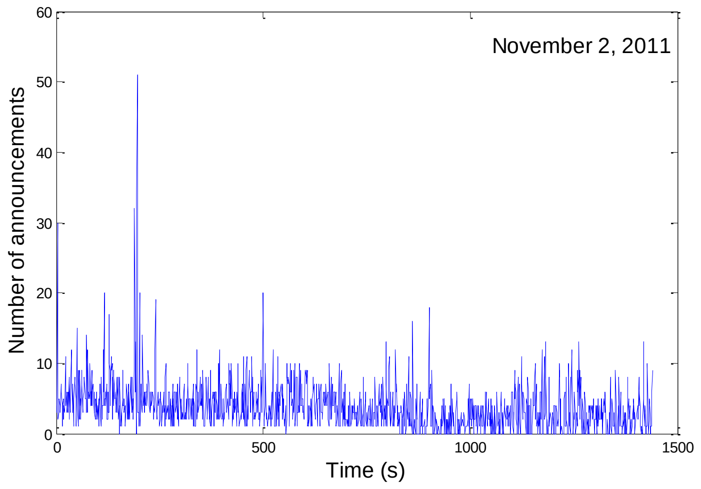


Figure 33. Number of Announcements on November 2, 2011.

The mean of announcements on December 2, 2011 and on October 2, 2011 are comparable. The statistics of number of announcements on October 2, 2011, November 2, 2011, and December 2, 2011 is shown in Table 12. However, the standard deviation on November 2, 2011 and December 2, 2011 are comparable. The range varies from 16 to 51 packets on the three chosen dates.

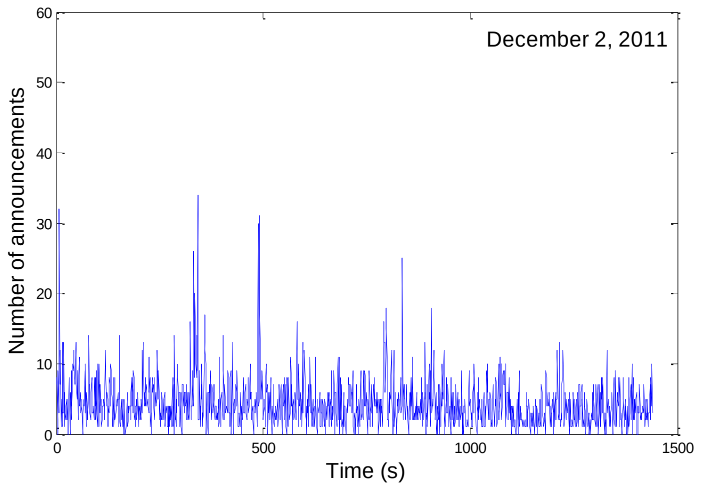


Figure 34. Number of Announcements on December 2, 2011.

Table 12. Statistics for Number of Announcements.

	Oct 2, 2011	Nov 2, 2011	Dec 2, 2011
Minimum	1	0	0
Maximum	16	51	34
Mean	4.72	3.84	4.36
Median	4	3	4
Mode	4	1	4
Standard Deviation	1.88	3.56	3.41
Range	15	51	34

6.2. Number of Withdrawals

The number of routes that are no longer reachable are called the number of BGP withdrawals. An *update* message may advertise only one route but several routes may be withdrawn. A BGP withdrawal requires simply the address of the network for which the route is being removed. The route is withdrawn from the RIB and if it is in the FIB, the route is removed and the route-selection algorithm chooses a new best route to that destination from the routes available in the RIB. NLRIs that have the same BGP attributes are encapsulated and sent to the BGP routers in one BGP packet [45]. Hence, one BGP packet may contain more than one announced or withdrawal NLRI prefix.

We notice a very small mean of only 1.11 packets in the number of withdrawals on October 2, 2011 throughout the 24-hour period, as shown in Figure 35. The number of withdrawals is related to the number of announcements. There were a few announcements on October 2, 2011. Hence, there were few withdrawals in the same time period. On November 2, 2011, the maximum value of withdrawals was 50 packets, as shown in Figure 36.

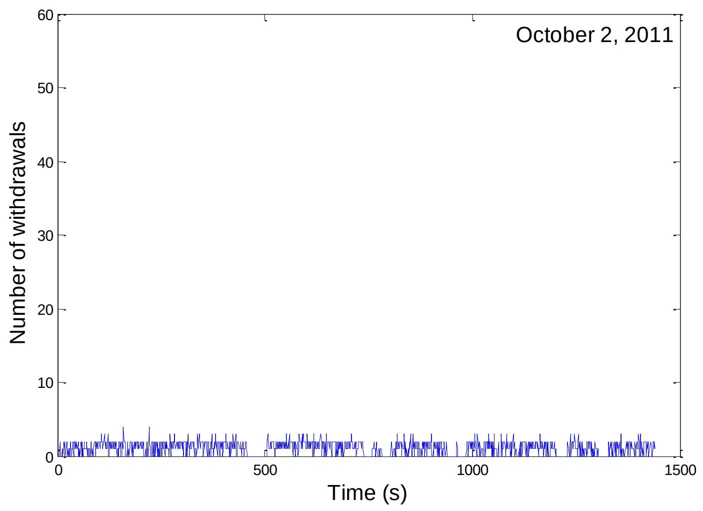


Figure 35. Number of Withdrawals on October 2, 2011.

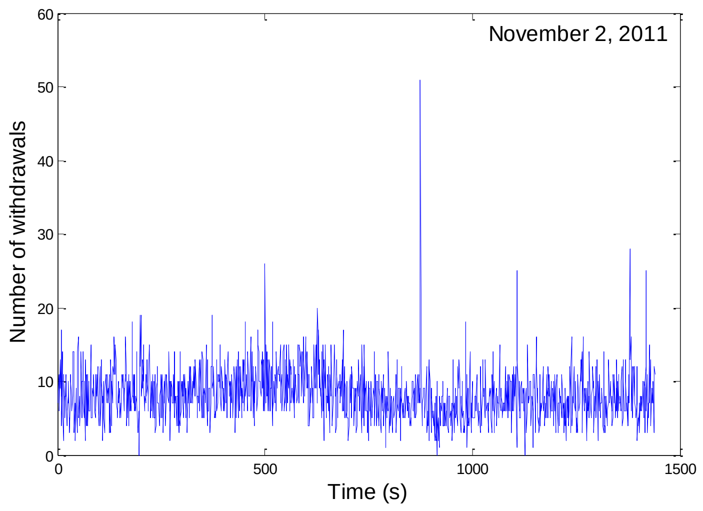


Figure 36. Number of Withdrawals on November 2, 2011.

The number of withdrawals on December 2, 2011 is shown in Figure 37. The mean of the number of announcements on November 2, 2011 and December 2, 2011 were approximately equal and the mean on October 2, 2011 was very small. This implies that a larger number of *update* messages was exchanged on November 2, 2011 and December 2, 2011 compared to October 2, 2011. The statistics for number of withdrawals on the three chosen dates is shown in Table 13.

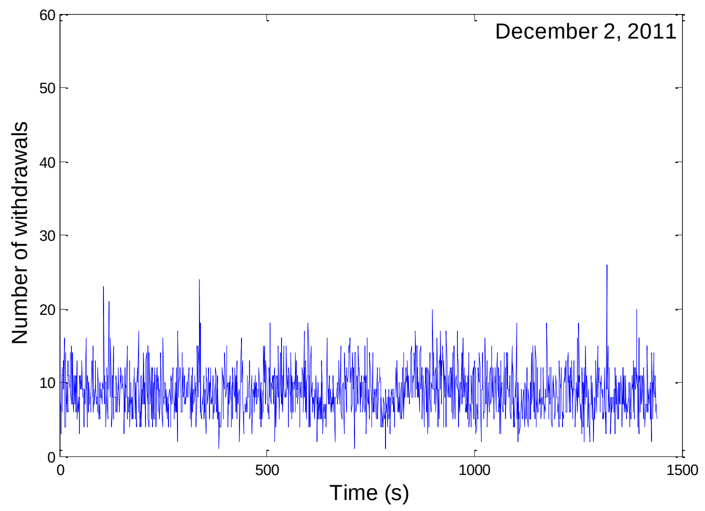


Figure 37. Number of Withdrawals on December 2, 2011.

Table 13. Statistics for Number of Withdrawals.

	Oct 2, 2011	Nov 2, 2011	Dec 2, 2011
Minimum	0	0	1
Maximum	4	50	26
Mean	1.11	8.23	8.74
Median	1	8	8
Mode	2	8	8
Standard Deviation	0.88	3.54	3.16
Range	4	50	25

6.3. Number of Announced Prefixes

BGP announces an IP prefix if a matching entry is found in the IP routing table. The number of announced prefixes on October 2, 2011 is shown Figure 38. To advertise a classless prefix, the prefix and the mask in the BGP routing process need to be configured. The number of announced prefixes represents the number of *update* Network Layer Reachability Information (NLRIs). The number of announced prefixes on November 2, 2011 is shown in Figure 39.

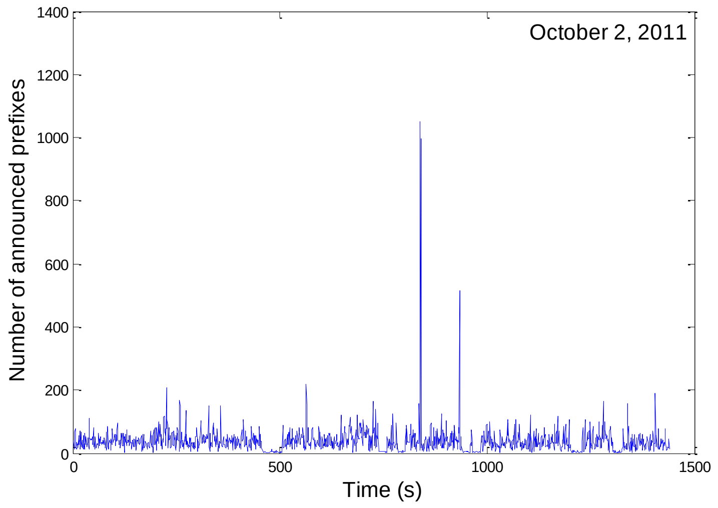


Figure 38. Number of Announced Prefixes on October 2, 2011.

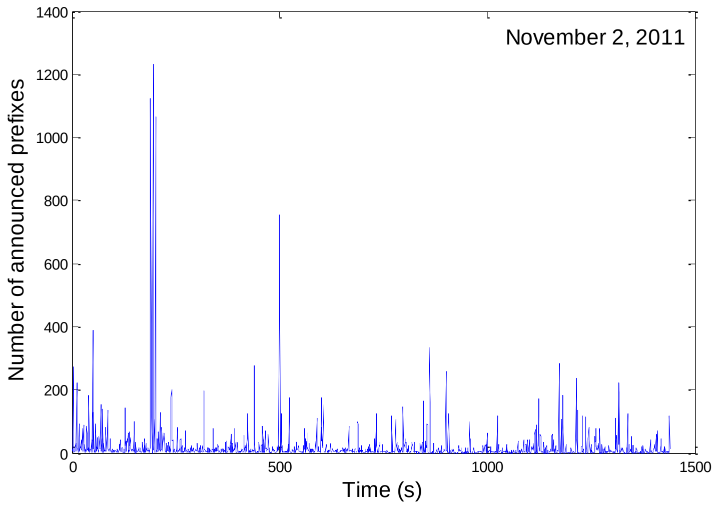


Figure 39. Number of Announced Prefixes on November 2, 2011.

The mean of announced prefixes was highest on October 2, 2011 when compared to mean of announced prefixes on November 2, 2011 and December 2, 2011. The number of announced prefixes on December is shown in Figure 40. The statistics for announced prefixes on the three chosen dates is shown in Table 14. The mean, median, mode, and standard deviation in all the three cases are distinct and are not comparable.

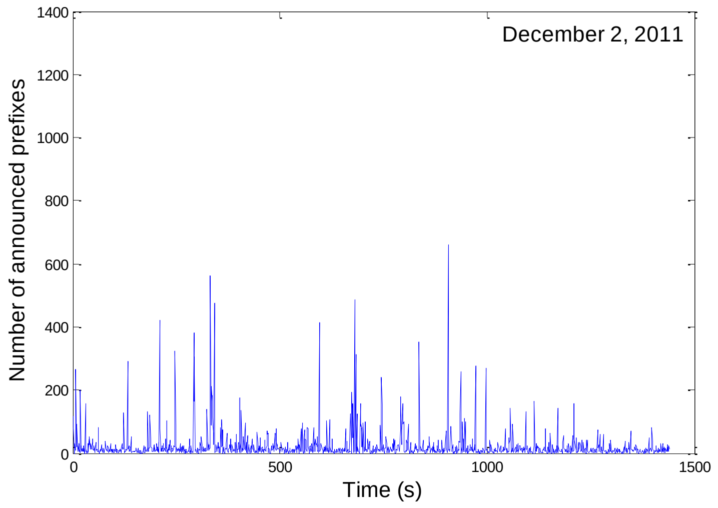


Figure 40. Number of Announced Prefixes on December 2, 2011.

Table 14. Statistics for Number of Announced Prefixes.

	Oct 2, 2011	Nov 2, 2011	Dec 2, 2011
Minimum	1	0	1
Maximum	1051	1233	26
Mean	37.82	20.72	8.74
Median	1	5	8
Mode	2	1	8
Standard Deviation	5.14	3.54	3.16
Range	1050	1233	25

6.4. Number of Withdrawn Prefixes

The number of prefixes that have been withdrawn over a period of time is known as the number of withdrawn prefixes. Additional information such as associated path attributes (AS Path) is not necessary for the routes being withdrawn. The number of withdrawn prefixes on October 2, 2011 is shown in Figure 41. The number of withdrawn prefixes on November 2, 2011 had a range of 1,335 as illustrated in Figure 42. A peak of 1,335 withdrawn prefixes was noticed on November 2, 2011, which was unusual.

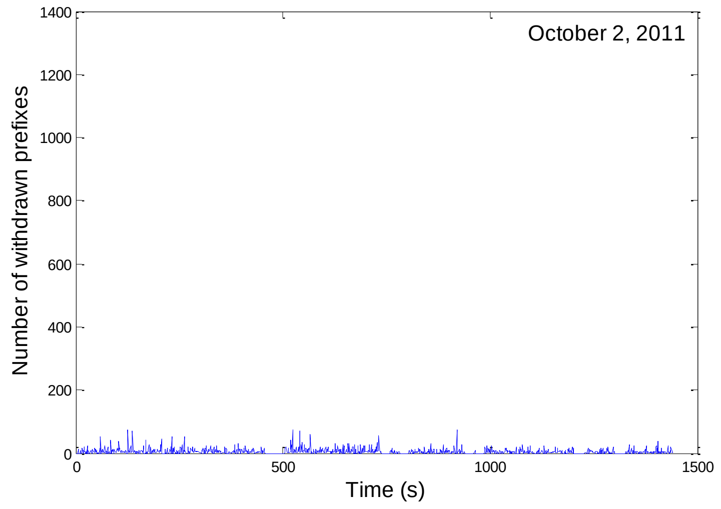


Figure 41. Number of Withdrawn Prefixes on October 2, 2011.

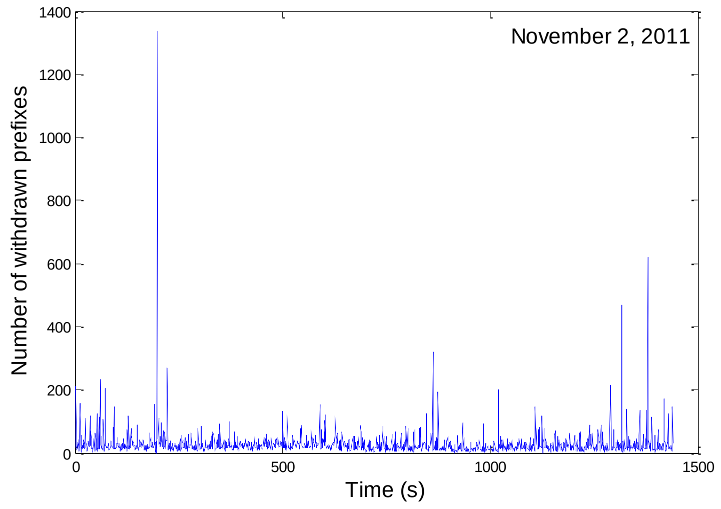


Figure 42. Number of Withdrawn Prefixes on November 2, 2011.

The number of withdrawn prefixes on December 2, 2011 is shown in Figure 43. The mean value of the number of withdrawn prefixes during this 24-hour period was 31.24 packets. Due to a short time interval of 24 hours, we were not able to detect any anomalies in this case. The mean of withdrawn prefixes on November 2, 2011 and December 2, 2011 are close to each other and therefore, are comparable. The mean of withdrawn prefixes on October 2, 2011 is very low. The statistics for withdrawn prefixes is shown in Table 15.

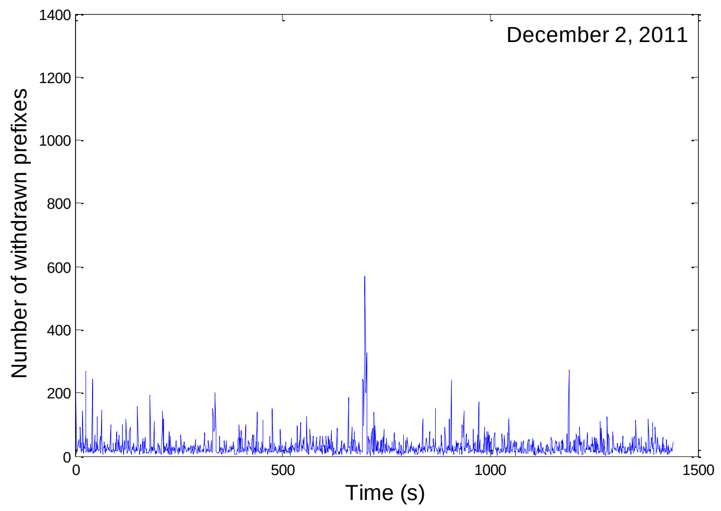


Figure 43. Number of Withdrawn Prefixes on December 2, 2011.

Table 15. Statistics for Withdrawn Prefixes.

	Oct 2, 2011	Nov 2, 2011	Dec 2, 2011
Minimum	0	0	2
Maximum	74	1335	571
Mean	5.93	28.35	31.24
Range	74	1335	569

6.5. Average AS Path

The average AS path is the average number of AS peers in the AS_path attribute of the BGP message. It is calculated from the packet flow and a unique pair of ASes to calculate the correlation between data flow and AS path length [49], [50]. The average AS path length is one of the factors that can be used to measure the interconnectedness of networks across the global Internet. The average AS path on October 2, 2011 and November 2, 2011 are shown in Figure 44 and Figure 45, respectively.

.

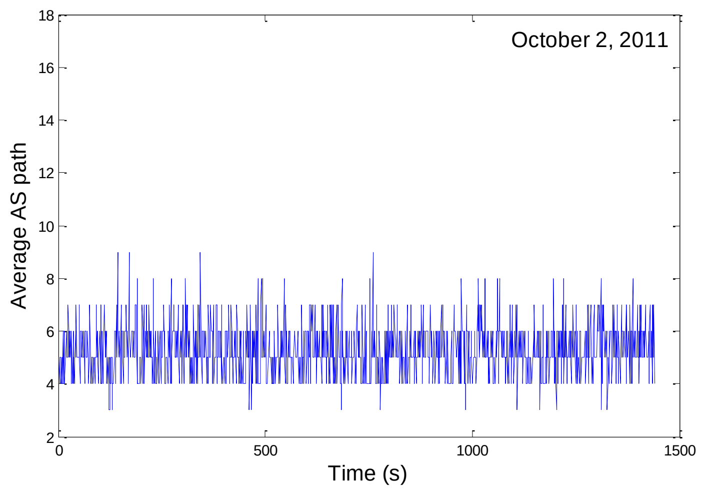


Figure 44. Average AS Path on October 2, 2011.

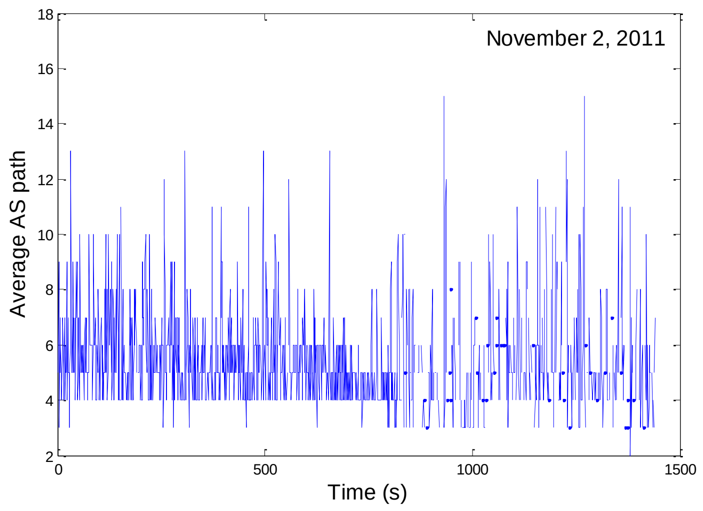


Figure 45. Average AS Path on November 2, 2011.

Average AS path had a range of 15 packets, as shown in Figure 46. The AS path length had a maximum value on December 2, 2011 and this may be due to the unavailability of shorter stable paths which should be preferred under normal circumstances. The statistics for average AS path on October 2, 2011, November 2, 2011, and December 2, 2011 are shown in Table 16.

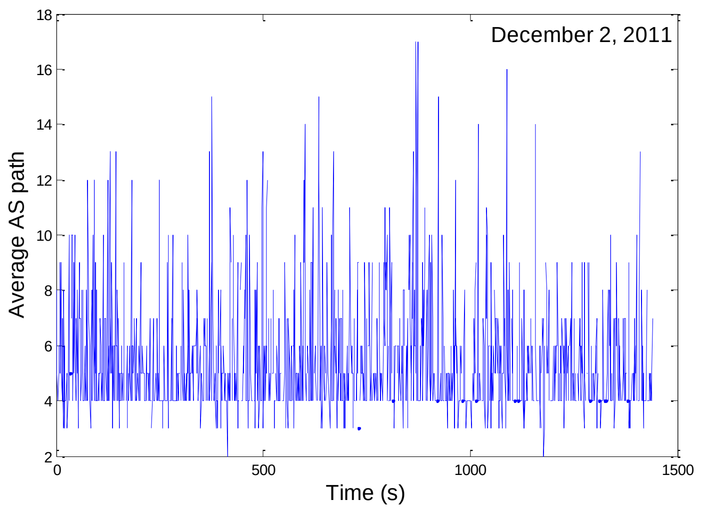


Figure 46. Average AS Path on December 2, 2011.

Table 16. Statistics for Average AS Path.

	Oct 2, 2011	Nov 2, 2011	Dec 2, 2011
Minimum	3	2	2
Maximum	9	15	17
Mean	5.93	28.35	31.24
Range	6	13	15

6.6. Maximum AS Path

The maximum number of AS peers during a one minute interval of BGP update messages is known as the maximum AS path. The BGP maximum AS (*maxas*-limit) length router configuration command is used to limit the maximum length of AS path. It reduces the impact of oversized AS-path attributes to the operation of a network. It had a range of 16 packets, as shown in Figure 47.

The mean of the maximum AS path in this particular case is 5.63 packets which is comparatively low compared to the mean of maximum AS path on October 2, 2011. The greater length of the AS_path attribute implies that the packet is routed via a longer path to the destination, which causes the observed long routing delays during BGP anomalies [34]. The maximum value of maximum AS path on November 2, 2011 is shown in Figure 48.

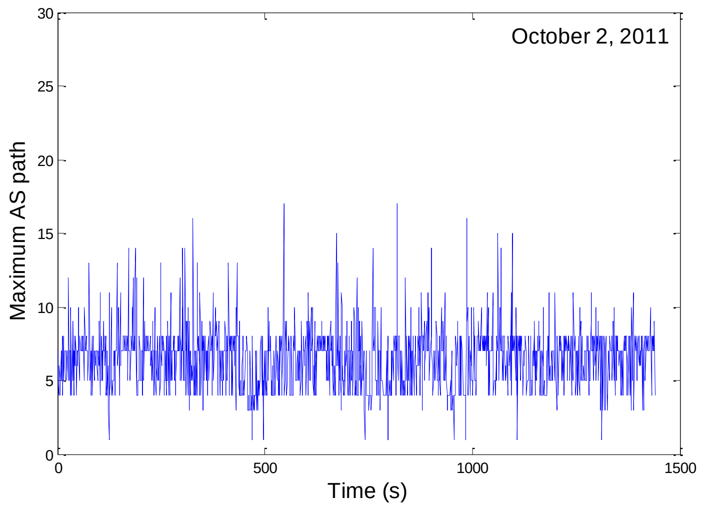


Figure 47. ***Maximum AS Path on October 2, 2011.***

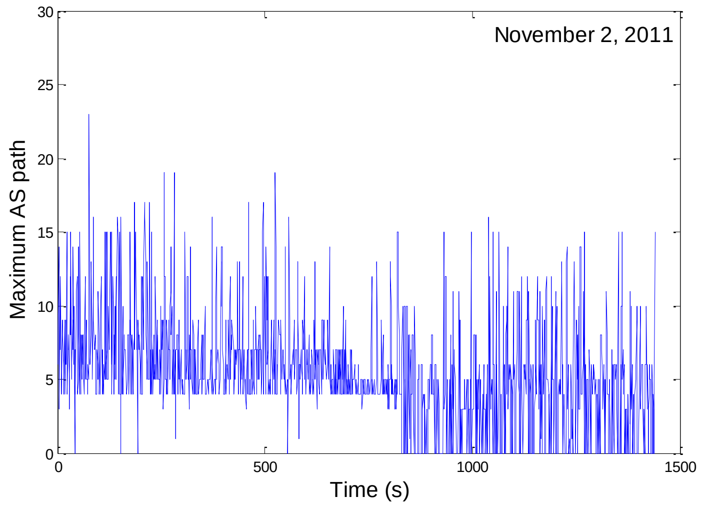


Figure 48. ***Maximum AS Path on November 2, 2011.***

The maximum AS path had maximum value of 26 packets on December 2, 2011, as shown in Figure 49. The mean in this case was 6.11 packets, which is close to mean of the maximum AS path on October 2, 2011. The statistics for maximum AS path on October 2, 2011, November 2, 2011, and December 2, 2011 are shown in Table 17.

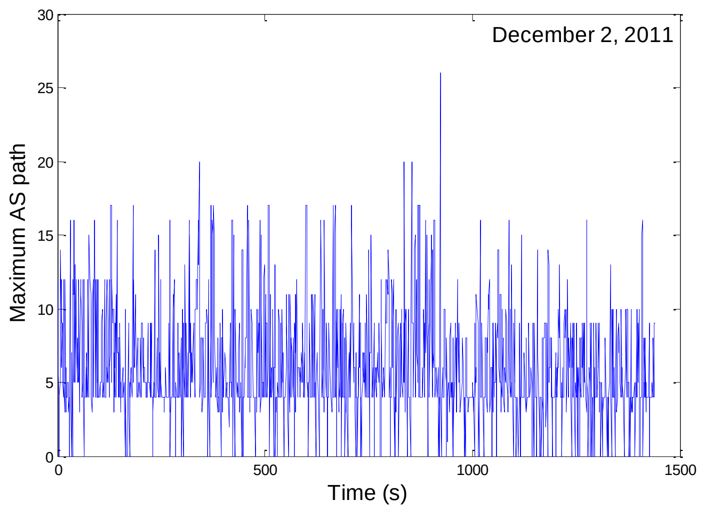


Figure 49. Maximum AS Path on December 2, 2011.

Table 17. Statistics for Maximum AS Path.

	Oct 2, 2011	Nov 2, 2011	Dec 2, 2011
Minimum	1	0	0
Maximum	17	23	26
Mean	6.49	5.63	6.11
Median	7	5	5
Mode	7	4	4
Standard Deviation	2.07	3.62	3.74
Range	16	23	26

6.7. Average Unique AS Path

The average unique AS path is the same as the average AS path length except that it considers the unique AS-path attributes during a one-minute period. The minimum and maximum values of average unique AS path on October 2, 2011 are shown in Figure 50. The average unique AS paths on November 2, 2011 is shown in Figure 51.

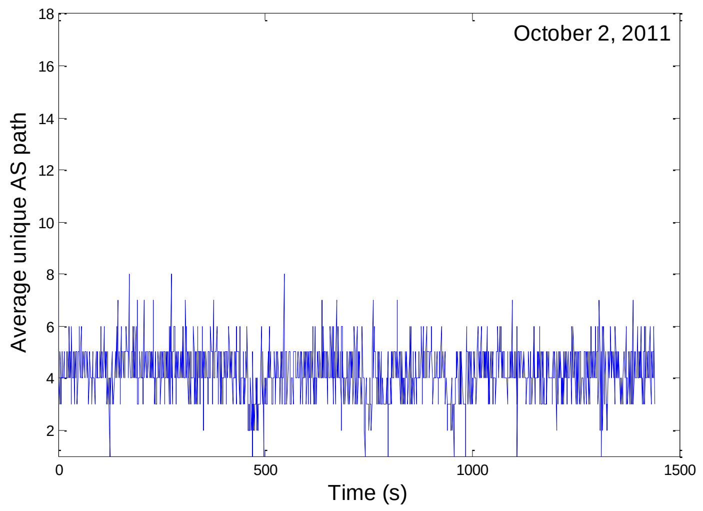


Figure 50. Average Unique AS Path on October 2, 2011.

.

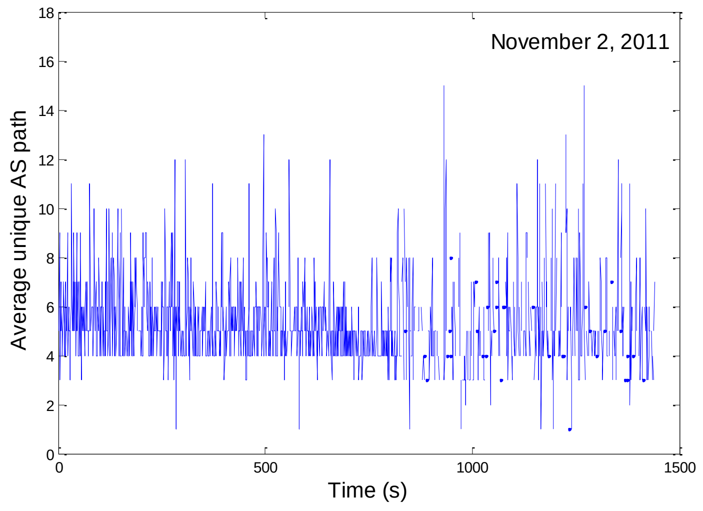


Figure 51. Average Unique AS Path on November 2, 2011.

The maximum values of unique AS path on November 2, 2011 and December 2, 2011 are comparable whereas, the maximum value of unique AS path is much lower. The graph depicting maximum and minimum values of unique AS paths is shown in Figure 52. The statistics of average unique AS path on the three selected dates is shown in Table 18.

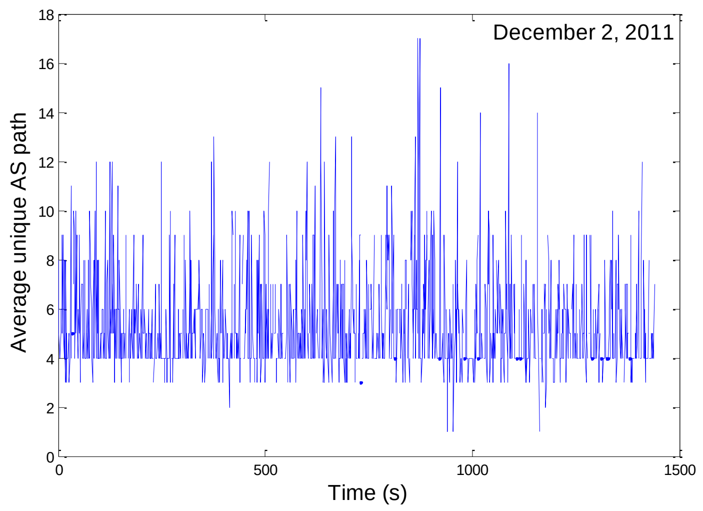


Figure 52. Average Unique AS Path on December 2, 2011.

Table 18. Statistics for Average Unique AS Path.

	Oct 2, 2011	Nov 2, 2011	Dec 2, 2011
Minimum	1	1	1
Maximum	8	15	17
Range	7	14	16

6.8. Duplicate BGP Announcements

A duplicate BGP announcement is an announcement that is identical to the last seen announcement for the same NLRI prefix and there is no change in either the AS-path or in any of the transitive route attributes. The duplicate announcements or withdrawals are responsible for the majority of router processing loads during their busiest times [51]. The duplicate BGP announcements on October 2, 2011 are shown in Figure 53. The mean of duplicate BGP announcements was 23.48 packets in this case, which is much lower than the mean of duplicate BGP announcements on November 2, 2011.

The minimum and maximum values of duplicate BGP announcements on November 2, 2011 are shown in Figure 54. The mean of duplicate BGP announcements in this particular case was 38.35 packets. On November 2, 2011 a maximum of 1,524 duplicate BGP announcements were noticed, this was unexpected compared to the duplicate BGP announcements throughout the 24-hour period. This may be due to link failures or routes that are no longer available.

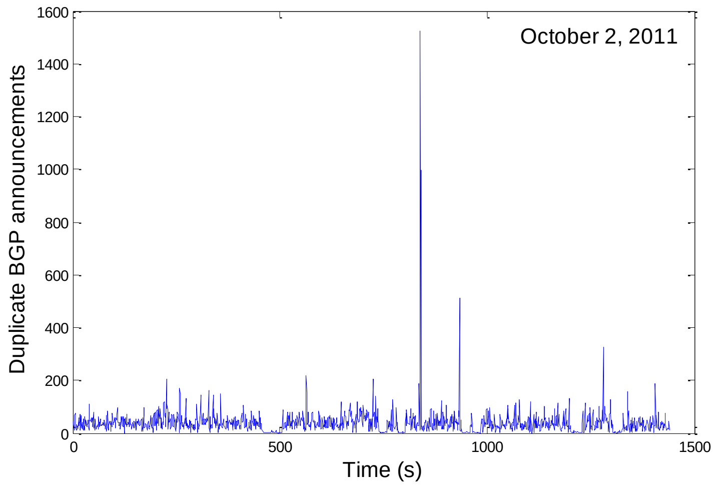


Figure 53. Duplicate BGP Announcements on October 2, 2011.

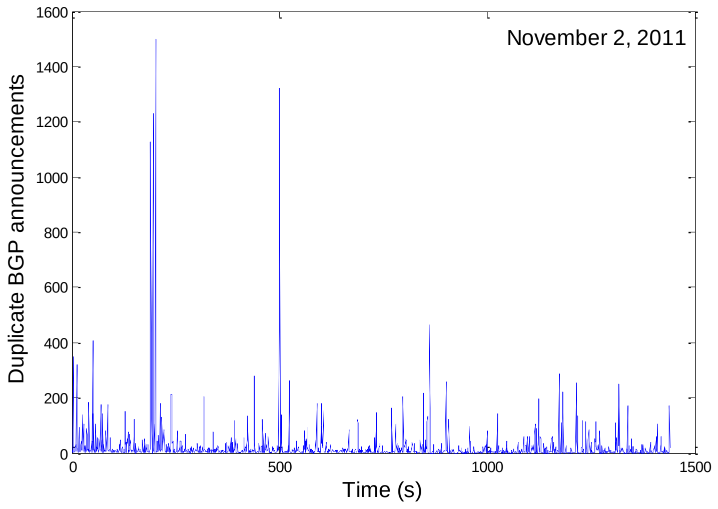


Figure 54. Duplicate BGP Announcements on November 2, 2011.

The maximum value on December 2, 2011 was 828 packets, which is much lower than the maximum values on October 2, 2011 and November 2, 2011, as shown in Figure 55. The mean in this particular case was 25.58 packets which is low compared to the mean on November 2, 2011 but close to mean on October 2, 2011. Table 19 shows the statistics for duplicate BGP announcements on the three chosen dates.

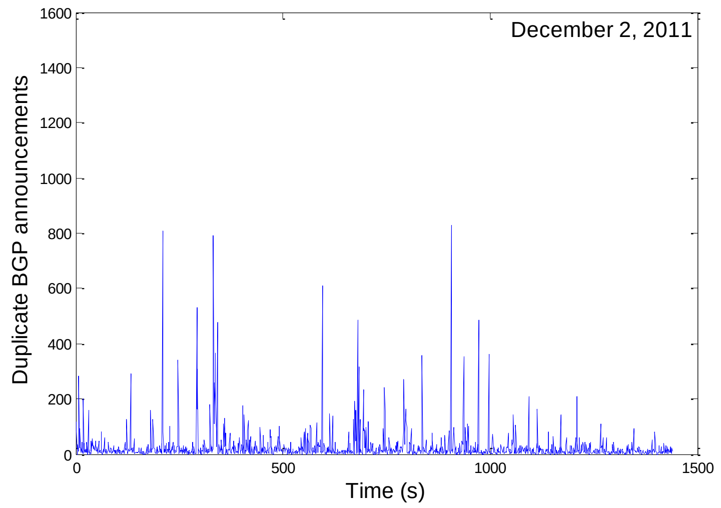


Figure 55. ***Duplicate BGP Announcements on December 2, 2011.***

Table 19. ***Statistics for Duplicate BGP Announcements.***

	Oct 2, 2011	Nov 2, 2011	Dec 2, 2011
Minimum	0	0	0
Maximum	1497	1524	828
Mean	23.48	38.35	25.58
Median	6	31	10
Mode	1	2	5
Standard Deviation	82.91	56.6	61.24
Range	1497	1524	828

6.9. Implicit Withdrawals

The number of times that a prefix has been withdrawn and re-advertised is called implicit withdrawal. This number is smaller than the total number of prefixes sent in that particular case. A BGP *update* message that re-announces a previously announced prefix with a different AS_path is considered to have implicitly withdrawn the earlier path [22]. The maximum value of implicit withdrawals on October 2, 2011 was 522 packets, as shown in Figure 56.

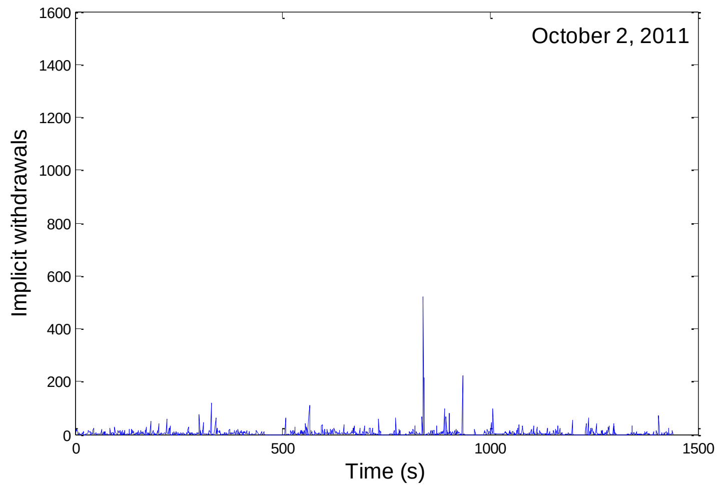


Figure 56. Implicit Withdrawals on October 2, 2011.

The implicit withdrawals had a maximum value of 1,542 packets on November 2, 2011, as shown in Figure 57. The mean of implicit withdrawals in this case was 8.49 packets, which is very high compared to the mean of implicit withdrawals on October 2, 2011.

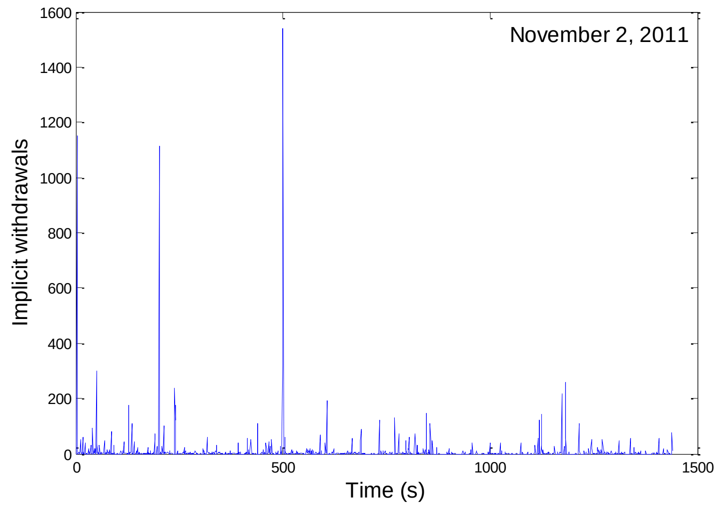


Figure 57. Implicit Withdrawals on November 2, 2011.

The maximum value of implicit withdrawals on December 2, 2011 was 925 packets, as shown in Figure 58, which was very low compared to the maximum value on November 2, 2011. The statistics of implicit withdrawals on the three chosen dates are shown in Table 20. The mean of implicit withdrawals on November 2, 2011 and

December 2, 2011 are closer to each other than the mean of implicit withdrawals on October 2, 2011.

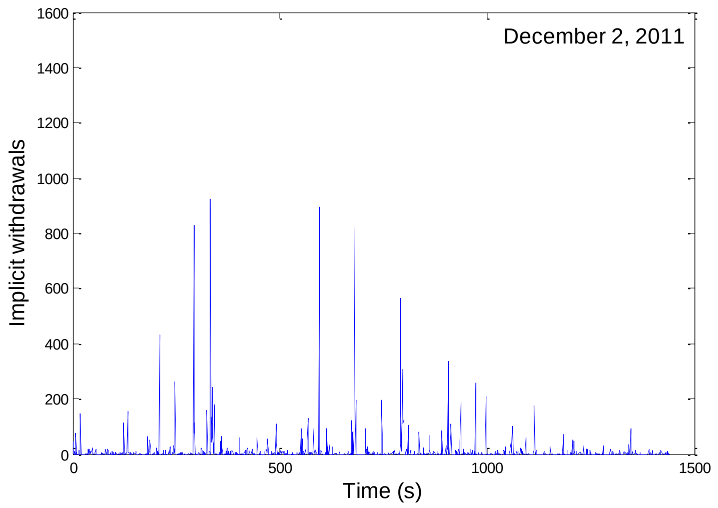


Figure 58. Implicit Withdrawals on December 2, 2011.

Table 20. Statistics for Implicit Withdrawals.

	Oct 2, 2011	Nov 2, 2011	Dec 2, 2011
Minimum	0	0	1
Maximum	522	1542	925
Mean	5.6	8.49	10.49
Standard Deviation	18.89	23.62	56.47
Range	522	1542	924

6.10. Duplicate BGP Withdrawals

A BGP withdrawal for a prefix sent by a router is a “duplicate” if all the attributes in the withdrawal are the same as the most recent previous withdrawal for prefix sent by router and both the update and the previous update belong to the same BGP session. The maximum value of duplicate BGP withdrawal on October 2, 2011 was 74 packets, as shown in Figure 59. The mean of the number of withdrawals on October 2, 2011 was very low. Hence, the mean of the duplicate BGP withdrawals is also very low.

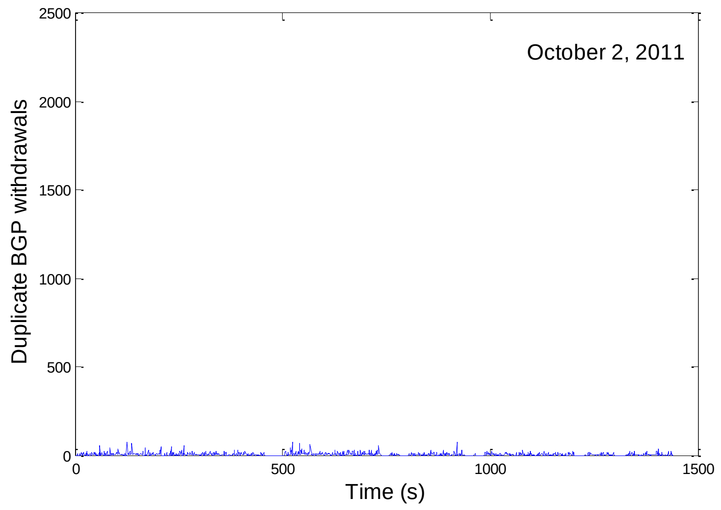


Figure 59. Duplicate BGP Withdrawals on October 2, 2011.

The duplicate BGP withdrawals had a maximum value of 1,701 packets on November 2, 2011, as shown in Figure 60, which is very high compared to the maximum value of duplicate BGP withdrawals on October 2, 2011. The duplicate BGP withdrawals on December 2, 2011 are shown in Figure 61.

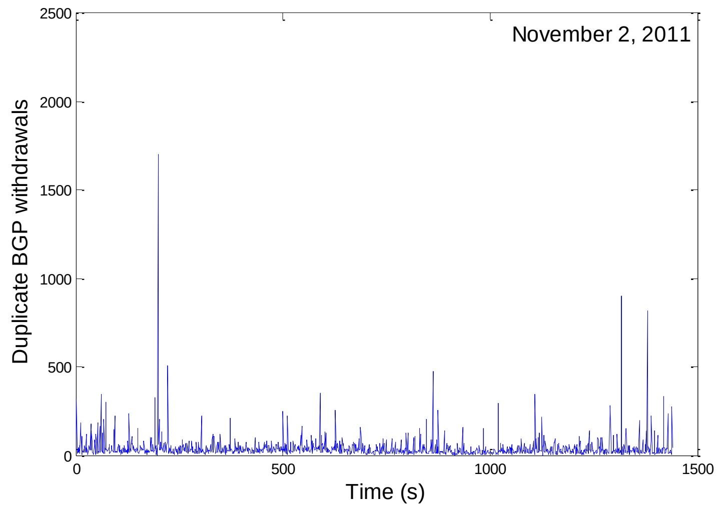


Figure 60. Duplicate BGP Withdrawals on November 2, 2011.

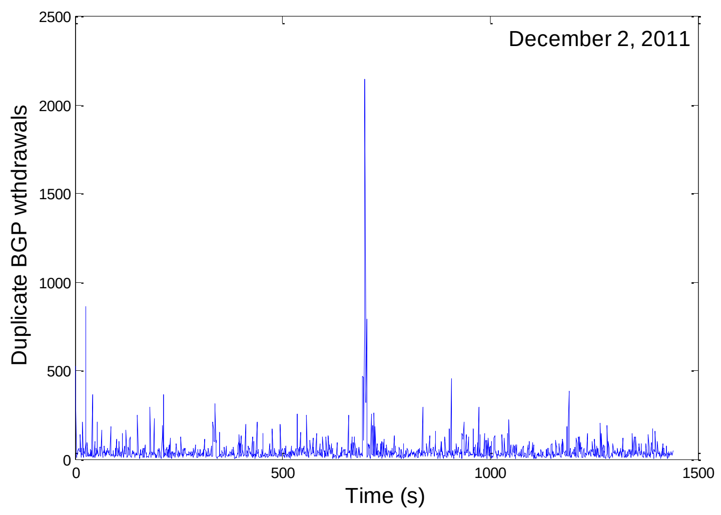


Figure 61. Duplicate BGP Withdrawals on December 2, 2011.

The mean of duplicate BGP withdrawals on October 2, 2011 is very low as compared to the mean on November 2, 2011 and December 2, 2011, which implies that there were very few duplicate BGP withdrawals found on October 2, 2011. The mean of duplicate BGP withdrawals is largest on December 2, 2011, which shows that additional duplicate BGP withdrawals were detected during this 24-hour period. Table 21 shows the statistics for duplicate BGP withdrawals on October 2, 2011, November 2, 2011, and December 2, 2011.

Table 21. Statistics for Duplicate BGP Withdrawals.

	Oct 2, 2011	Nov 2, 2011	Dec 2, 2011
Minimum	0	0	2
Maximum	74	1701	2145
Mean	5.14	39.49	49.45
Median	2	25	30
Standard Deviation	8.26	69.53	100.4
Range	74	1701	2143

6.11. Maximum AS Path Edit Distance

The edit distance between two AS path attributes is defined as the minimum number of insertions, deletions, and substitutions that need to be executed to match the attributes. The value of the edit distance feature was extracted by computing the edit distance among the AS path attributes at each time interval [52]. The maximum value of maximum AS path edit distance on October 2, 2011 was 11 packets, as shown in Figure 62.

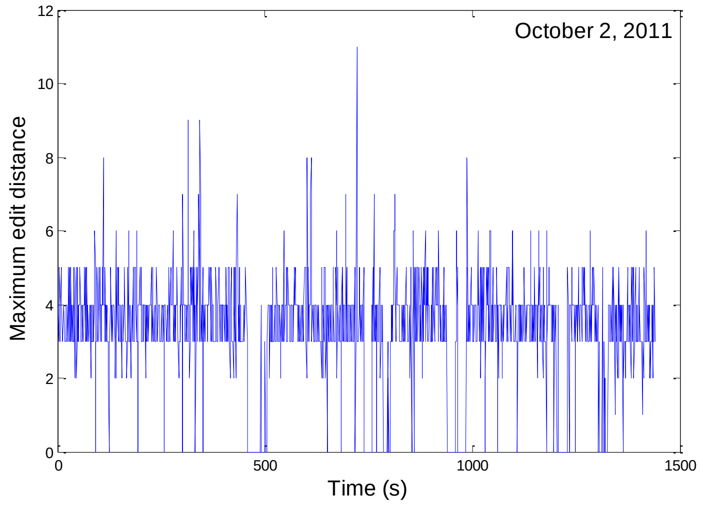


Figure 62. Maximum AS Path Edit Distance on October 2, 2011.

The maximum values of maximum AS path edit distance on November 2, 2011 and December 2, 2011 were 11 packets, as shown in Figure 63 and Figure 64. The mean of the maximum AS path edit distance on the three dates was very close to each other. The maximum value was the same in these three cases. The statistics for maximum AS path edit distance on October 2, 2011, November 2, 2011, and December 2, 2011 are shown in Table 22.

Table 22. Statistics for Maximum AS Path Edit Distance.

	Oct 2, 2011	Nov 2, 2011	Dec 2, 2011
Minimum	0	0	0
Maximum	11	11	11
Mean	3.25	2.82	3.08
Median	3	4	4
Standard Deviation	1.53	2.34	2.25
Range	11	11	924

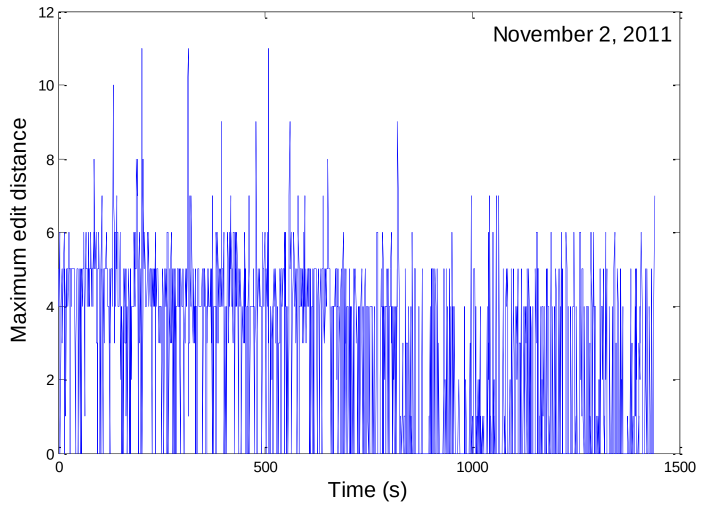


Figure 63. Maximum AS Path Edit Distance on November 2, 2011.

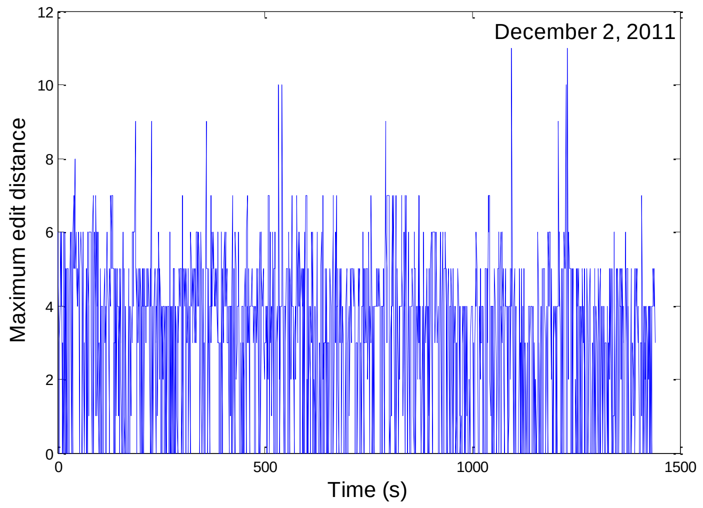


Figure 64. Maximum AS Path Edit Distance on December 2, 2011.

6.12. Average AS Path Edit Distance

The value of edit distance attribute was extracted by computing the edit distance among the AS path attributes during a one-minute time interval [32]. The Levenshtein distance is used for measuring the difference between two sequences. The term 'edit distance' is often used to refer particularly to Levenshtein distance [53], [54]. For example, the Levenshtein distance between "home" and "gone" is 2: two edits change one into the other and there is no way of doing it with fewer than two edits:

- **h**ome → **g**ome (substitution of 'g' for 'h')
- go**m**e → go**n**e (substitution of 'n' for 'm').

The edit distance is computed by finding the sequence of string edit operations. The number of insertions, deletions, or substitutions needed to convert one string to other is known as the edit distance between those two strings. The C# code for computing edit distance between two AS paths is given in Appendix B.

To compute the Levenshtein distance, reserve a matrix to hold the Levenshtein distances between all prefixes of the first string and all prefixes of the second and then compute the values in the matrix and in the end find the distance between the two full strings as the last value computed.

The Levenshtein distance between two strings a, b is given by $lev_{a,b}(|a|,|b|)$

where

$$lev_{a,b}(i,j) = \left\{ \begin{array}{lr} 0, & i = j = 0 \\ i, & j = 0 \; or \; i > 0 \\ j, & i = 0 \; or \; j > 0 \\ \min \left\{ \begin{array}{c} lev_{a,b}\,(i-1,j)+1 \\ lev_{a,b}\,(j,1-1)+1 \\ lev_{a,b}(i-1,j-1)+[a \neq b], \;\; else \end{array} \right\} & \end{array} \right\}.$$

The first element in the minimum corresponds to insertion (from a to b), the second to deletion, and the third to substitution.

The maximum and minimum values of average AS path edit distance on October 2, 2011, November 2, 2011, and December 2, 2011 were the same, as shown in Figure 65, Figure 66, and Figure 67. The mean of the average AS path edit distance on these three dates were 2, 1, and 1 packet, respectively.

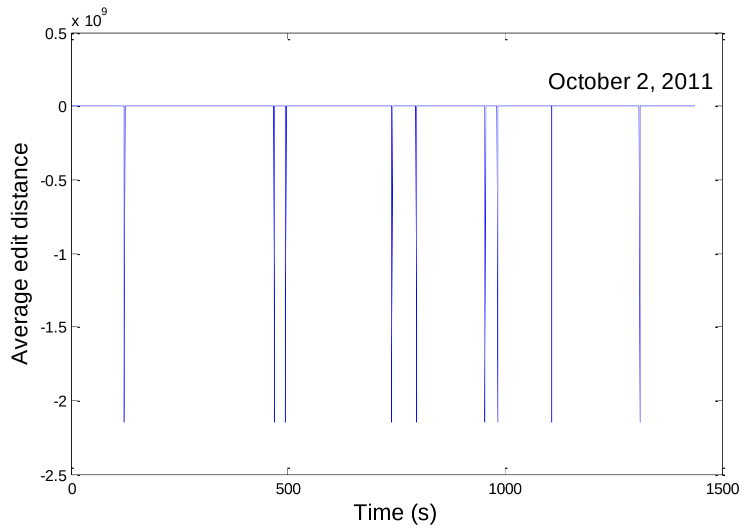


Figure 65. Average AS Path Edit Distance on October 2, 2011.

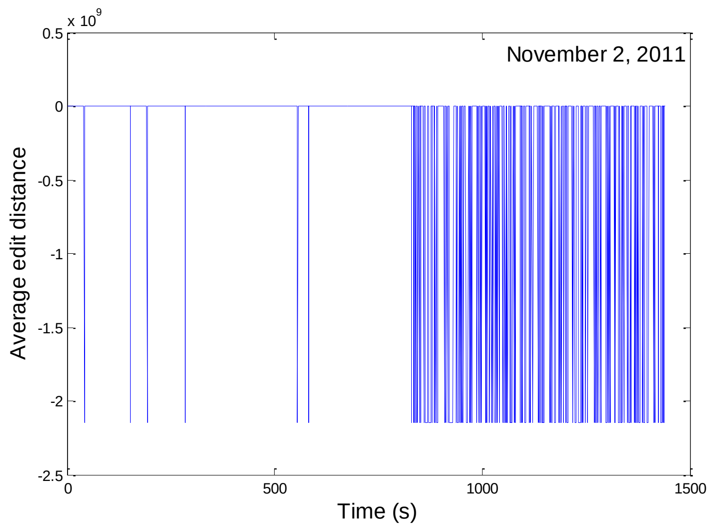


Figure 66. Average AS Path Edit Distance on November 2, 2011.

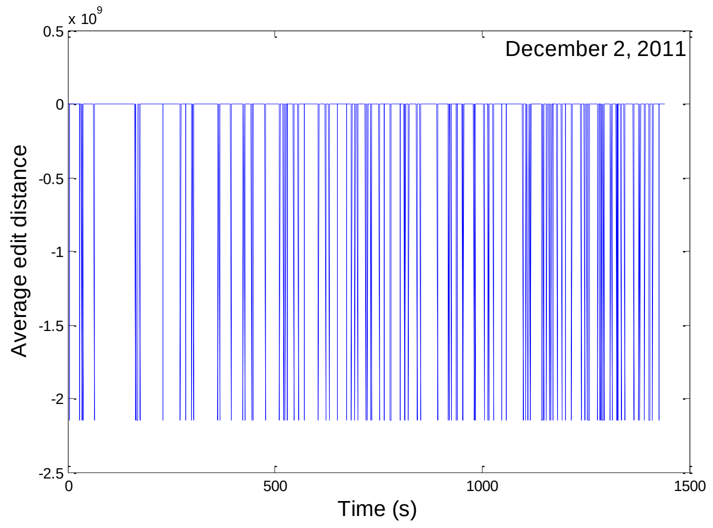


Figure 67. Average AS Path Edit Distance on December 2, 2011.

6.13. Number of IGP Packets

The number of routing protocols that are used to exchange routing information within an AS is called the number of IGP packets. The interior gateway protocols can be divided into two categories: a) distance-vector routing protocol and b) link-state routing protocol. The minimum and the maximum values of number of IGP packets on October 2, 2011 were 1 and 16 packets, respectively, as shown in Figure 68. The number of IGP packets on October 2, 2011 had a very low mean of 4.74.

The maximum and the minimum values of the number of IGP packets on December 2, 2011 were 156 and 31, respectively, as shown in Figure 70. The statistics of IGP packets on October 2, 2011, November 2, 2011, and December 2, 2011 are shown in Table 23.

Table 23. Statistics for IGP Packets.

	Oct 2, 2011	Nov 2, 2011	Dec 2, 2011
Minimum	1	1	31
Maximum	16	139	156
Mean	4.74	56.92	57.19
Median	4	56	56
Mode	4	60	52
Standard Deviation	1.9	11.65	11.03
Range	15	126	125

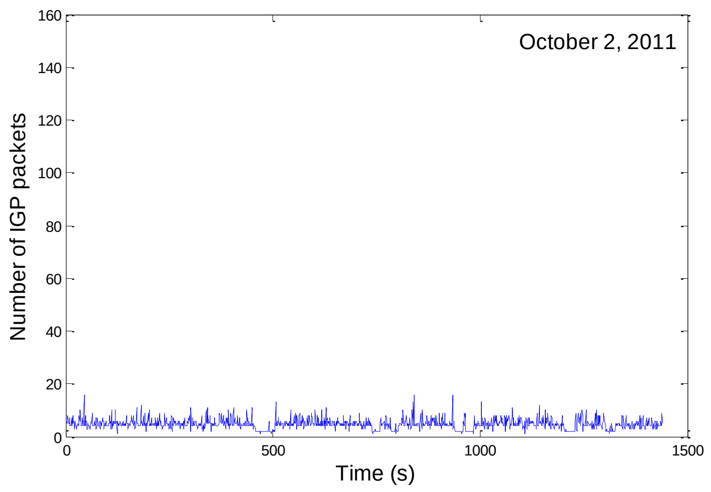


Figure 68. Number of IGP Packets on October 2, 2011.

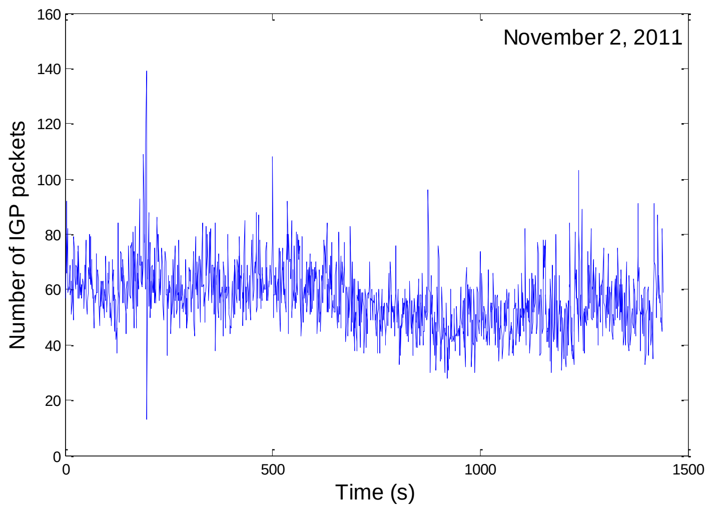


Figure 69. Number of IGP Packets on November 2, 2011.

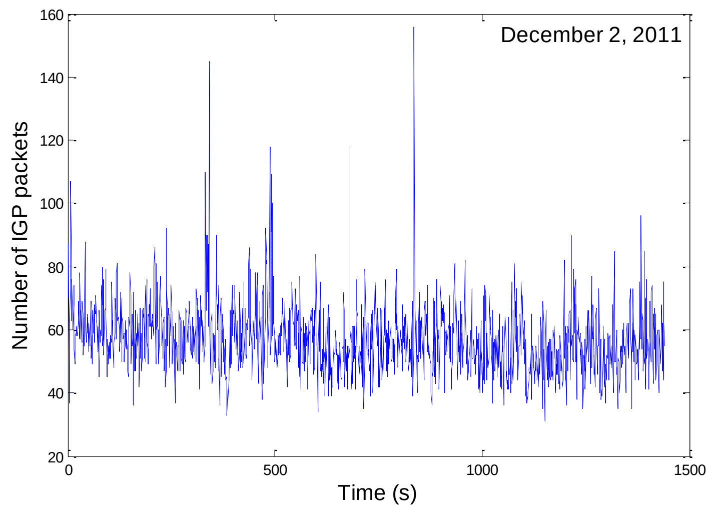


Figure 70. Number of IGP Packets on December 2, 2011.

6.14. Average Packet Size

The average packet size is calculated as the size of the packets divided by the total number of packets. The periodic stream of average packet size on October 2, 2011 over a long period of time has a "clothesline" phenomenon [2], as shown in Figure 71. This may be due to route flapping [3]. A route "flaps" when it exhibits routing oscillations. RFD mechanisms are employed by the BGP to prevent persistent routing oscillations caused by network instabilities such as router configuration errors, transient data link failures, and software defects [55].

The mode in case of October 2, 2011 was very low compared to mode of average packet size on November 2, 2011 and December 2, 2011. The mean of average packet size on November 2, 2011 and December 2, 2011 was 5,087 and 8,351 packets, respectively, as shown in Figure 72 and Figure 73.

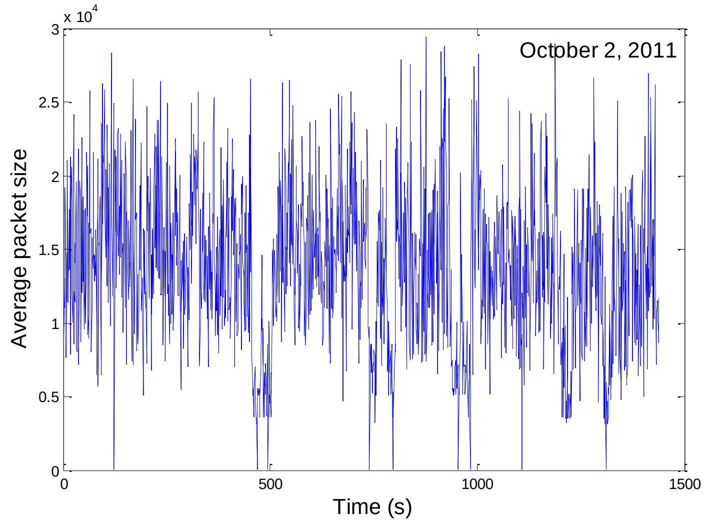


Figure 71. Average Packet Size on October 2, 2011.

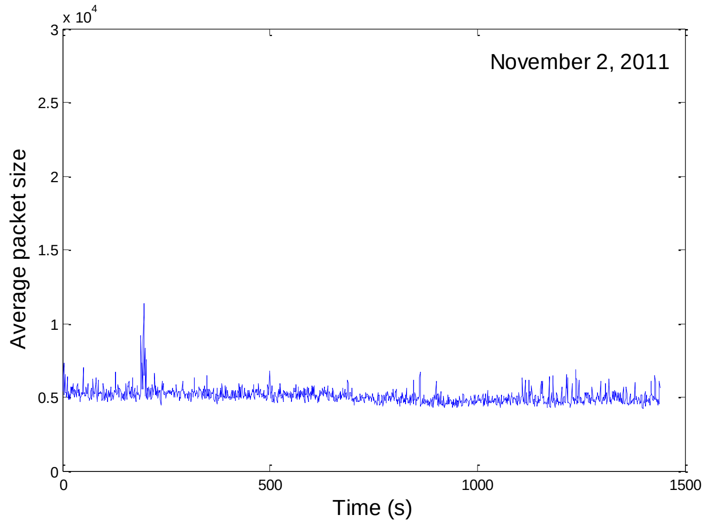
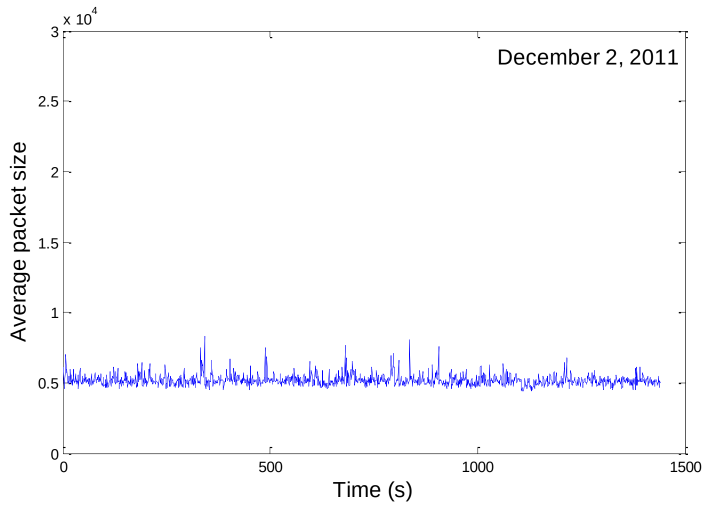


Figure 72. Average Packet Size on November 2, 2011.



.

Figure 73. Average Packet Size on December 2, 2011.

7. Future Work

Future work may involve performance analysis of the BGP protocol and its dependence on various algorithms and parameters such as route flap damping and minimal route advertisement interval. Conditions such as worm attacks, link outages, and router failure lead to route fluctuations that affect the quality of service of the Internet. Therefore, it is necessary to detect and limit the routing instabilities or anomalies [25]. The data collected from BCNET may be compared to datasets publicly available from repositories such as Route Views and RIPE.

The extracted attributes may be clustered based on the similarities in their properties. There are many types of clustering techniques such as k-means, hierarchical clustering, and DBSCAN. Tools such as RapidMiner [56] and Weka [57] may be used for this purpose. Formal verification of BGP specification validates whether or not a specific set of requirements is satisfied. In recent years, the probabilistic behavior of BGP has been explored. A probabilistic model-checking approach may be used to analyze the safety of a BGP policy and the BGP convergence time using Probabilistic Computation Tree Logic (PCTL) [58].

The features extracted using C# code may be used to detect and classify BGP anomalies such as IP prefix hijack, worms, mis-configurations, and electricity failures that affect the global internet BGP routing. Statistical and machine learning techniques may be used to classify and detect BGP anomalies. These anomalies can be detected by various techniques such as BGP lens [59], Support Vector Machines (SVMs) [60], and Hidden Markov Models (HMMs) [61] and then classified accordingly.

8. Conclusions

In this thesis, we collected BCNET BGP traffic traces. We provided details of the special purpose hardware used for data collection. The main objective of this project was to extract some useful data from the raw data and examine to further improve the routing protocol. This was the first step in analyzing data with free software tools such as Wireshark packet analyzer and Walrus visualization tool. The testbed and BCNET measurements were described.

We observed different types of BGP messages and considered BGP *update* messages for the purpose of analysis. We used a tool written in C# to parse and extract the desired attributes. *Update* messages consisted of announcement and withdrawal messages for NLRI prefixes. We considered different attributes on three dates and compared them. These attributes included number of announcements, number of withdrawals, number of announced NLRI prefixes, number of withdrawn NLRI prefixes, average AS path length, maximum AS path length, average unique AS path length, number of duplicate announcements, number of duplicate withdrawals, number of implicit withdrawals, average edit distance, maximum edit distance, number of EGP packets, and packet size.

There was only one transit provider (Tata) recognized for the data collected in October, November, and December 2011. There should be at least three transit providers. However, since BCNET was in a period of transition to convert all 1-Gig service providers to 10-Gig service providers therefore, we had only one service provider during that period. The BCNET data points collected between December 20 and December 22, 2010 contained no anomalies. We used Wireshark to import and export data packets and we analyzed the data and created various statistics. One disadvantage of using Wireshark was that we could not detect the problems if there are any but it might warn us about possible problems.

References

[1] Y. Rekhter, T. Li, and S. Hares, “A border gateway protocol 4 (BGP-4),” *IETF RFC 1771.*

[2] B. A. Prakash, N. Valler, D. Andersen, M. Faloutsos, and C. Faloutsos, “BGP-lens: Patterns and anomalies in Internet routing updates,” in *Proc. ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, Paris, France, June 2009, pp. 1315–1324.

[3] W. Shen and Lj. Trajkovic, “BGP route flap damping algorithms,” in *Proc. SPECTS 2005*, Philadelphia, PA, July 2005, pp. 488–495.

[4] N. Laskovic and Lj. Trajkovic, “BGP with an adaptive minimal route advertisement interval,” in *Proc. 25th IEEE Int. Performance, Computing, and Communications Conference*, Phoenix, AZ, April 2006, pp. 135–142.

[5] BGP datasets [Online]. Available: http://archive.routeviews.org.

[6] Reseaux IP Europeens [Online]. Available: http://www.ripe.net/ris.

[7] W. Leland, M. Taqqu, W. Willinger, and D. Wilson, “On the self-similar nature of Ethernet traffic (extended version),” in Proc. *IEEE/ACM Trans. Networking*, vol. 2, pp. 1–15, February 1994.

[8] M. Jiang, M. Nikolic, S. Hardy, and Lj. Trajkovic, “Impact of self-similarity on wireless network performance,” in *Proc. IEEE Int. Conf. on Communications, ICC 2001*, Helsinki, Finland, June 2001, pp. 477–481.

[9] J. Agosta and T. Russell, CDPD: Cellular Digital Packet Data Standards and Technology. Reading, MA: McGrawHill, 1996.

[10] I. Katzela, Modeling and Simulating Communication Networks: *A Hands-On Approach Using OPNET*. Upper Saddle River, NJ: Prentice Hall, 1999.

[11] Chinasat [Online]: Available: http://en.wikipedia.org/wiki/Chinasat.

[12] S. Lau and Lj. Trajkovic, “Analysis of traffic data from a hybrid satellite-terrestrial network,” in *Proc. The Fourth Int. Conf. on Quality of Service in Heterogeneous Wired/Wireless Networks (QShine 2007)*, Vancouver, BC, Canada, August 2007.

[13] D. Sharp, N. Cackov, N. Laskovic, Q. Shao, and Lj. Trajkovic, “Analysis of public safety traffic on trunked land mobile radio systems,” *IEEE J. Select. Areas Commun.*, vol. 22, no. 7, pp. 1197–1205, September 2004.

[14] E-Comm: Emergency Communications for Southwest British Columbia [Online]. Available: http://www.ecomm.bc.ca.

[15] B. Vujicic, H. Chen, and Lj. Trajkovic, “Prediction of traffic in a public safety network,” in *Proc. IEEE Int. Symp. Circuits and Systems*, Kos, Greece, May 2006, pp. 2637–2640.

[16] G. Siganos, M. Faloutsos, P. Faloutsos, and C. Faloutsos “Power-laws and the AS-level Internet topology,” *IEEE/ACM Trans. Networking*, vol.11, no. 4, pp. 514–524, August 2003.

[17] L. Subedi and Lj. Trajkovic, “Spectral analysis of Internet topology graphs,” in *Proc. IEEE Int. Symp. Circuits and Systems*, Paris, France, June 2010, pp. 1803–1806.

[18] I. Trestian, S. Ranjan, A. Kuzmanovic, and A. Nucci, “Googling the Internet: profiling internet endpoints via the world wide web,” *IEEE/ACM Transactions on Networking* vol. 18, no. 2, pp. 666–679, April 2010.

[19] BGP monitoring and analyzer tool: BGPmon [Online]. Available: http://www.bgpmon.net.

[20] BGP Data Analysis Project: BDAP [Online]. Available: http://web2.clarkson.edu/projects/itl/HOWTOS/bgpAnalysis/.

[21] D. Blazakis, M. Karir, and J.S. Baras, “BGP-Inspect: Extracting information from raw BGP data,” *in Proc. IEEE/IFIP Network Operations and Management Symposium*, Vancouver, BC, Canada, April 2006.

[22] BGP Routing Table Analysis [Online]. Available: http://www.potaroo.net/tools/asns/.

[23] G. Huston, “Analyzing the Internet's BGP routing table,” *The Internet Protocol Journal*, vol. 4, no. 1, March 2001, http://www.potaroo.net/papers/2001-3-bgptable/4-1-bgp.pdf.

[24] Autonomous System Numbers [Online]. Available: http://www.iana.org/assignments/as-numbers.

[25] A. Elmokashfi, A. Kvalbein, and C. Dovrolis, “On the scalability of BGP: the role of topology growth,” *IEEE Journal on Selected Areas in Communications, Special issue: Internet Routing Scalability,* October 2010, pp.1250–1261.

[26] RFC 1771 [Online]. Available: http://www.rfc-editor.org/rfc/rfc1771.txt.

[27] BCNET [Online]. Available: http://www.bc.net.

[28] BCNET Traffic Map [Online].
Available: https://www.bc.net/atlconf/display/Network/BCNET+Traffic+Map.

[29] Data Monitoring Switch [Online].
Available: http://www.netoptics.com/products/director.

[30] S. Lally, T. Farah, R. Gill, R. Paul, N. Al-Rousan, and Lj. Trajkovic, "Collection and characterization of BCNET BGP traffic," in *Proc. 2011 IEEE Pacific Rim Conference on Communications, Computers and Signal Processing,* Victoria, BC, Canada, August 2011, pp. 830–835.

[31] Wireshark [Online]. Available: http://www.wireshark.org.

[32] Wireshark User's Guide [Online]. Available: http://www.wireshark.org/docs/wsug_html_chunked/ChAdvTimestamps.html.

[33] OpenFabrics Alliance Archive [Online]. Available: http://www.openfabrics.org/archives/spring2008sonoma/Wednesday/Endace-Wednesday.ppt.

[34] Welcome to DAG [Online]. Available: http://www.endace.com.

[35] D. L. Mills, "Exterior Gateway Protocol formal specification," *IETF RFC 904*.

[36] BGP Case Studies [Online]. Available: http://www.cisco.com/en/US/tech/tk365/technologies_tech_note09186a008 00c95bb.shtml.

[37] V. Paxson and M. Allman, "Computing TCP's retransmission timer," *IETF RFC 2988*.

[38] W. Feng and P. Tinnakornsrisuphap, "The adverse impact of the TCP congestion-control mechanism in heterogeneous computing systems," in *Proc. The International Conference on Parallel Processing*, Toronto, Canada, August 2000, pp. 299–306.

[39] J. F. Kurose and K. W. Ross, "*Transport layer*," in *Computer Networking: A Top-down Approach*, 4th ed, New York: Pearson International, 2007, pp. 307–308.

[40] J. Zhang, J. Rexford, and J. Feigenbaum, "Learning-based anomaly detection in BGP updates," in *Proc. ACM SIGCOMM Workshop on Mining Network Data*, Philadelphia, PA, USA, August 2005, pp. 219–220.

[41] D. Moore, V. Paxson, S. Savage, C. Shannon, S. Staniford, and N. Weaver, "Inside the Slammer worm," *IEEE Security and Privacy,* vol. 1, no. 4, pp. 33–39 July 2003.

[42] A. Machie, J. Roculan, R. Russell, and M. V. Velzen, "Nimda worm analysis," Tech. Rep., *Incident Analysis*, Security Focus, September 2001.

[43] D. Moore, C. Shannon, and J. Brown, “Code-Red: a case study on the spread and victims of an Internet worm,” in *Proc. 2nd ACM SIGCOMM Internet Measurement Workshop*, Marseille, France, November 2002, pp. 273–284.

[44] N. Al-Rousan and Lj. Trajkovic, “Comparison of machine learning models for classification of BGP anomalies,” in *Proc. HPSR 2012*, Belgrade, Serbia, June 2012, pp. 103-108.

[45] N. Al-Rousan, S. Haeri, and Lj. Trajkovic, “Feature selection for classification of BGP anomalies using Bayesian models,” in *Proc. ICMLC 2012*, Xi'an, China, July 2012.

[46] Cooperative Association for Internet Data Analysis [Online]. Available: http://www.caida.org.

[47] Walrus - Graph Visualization Tool [Online]. Available: http://www.caida.org/tools/visualization/walrus.

[48] T. Farah, S. Lally, R. Gill, N. Al-Rousan, R. Paul, D. Xu, and Lj. Trajkovic, “Collection of BCNET BGP traffic,” in *Proc. 23rd International Teletraffic Congress,* San Francisco, CA, USA, September 2011, pp. 322–323.

[49] D. Meyer, “BGP communities for data collection,” RFC 4384, *IETF*, 2006 [Online]. Available: http://www.ietf.org/rfc/rfc4384.txt.

[50] S. Deshpande, M. Thottan, T. K. Ho, and B. Sikdar, “An online mechanism for BGP instability detection and analysis,” *IEEE Trans. Computers*, vol. 58, no. 11, pp. 1470–1484, November 2009.

[51] L. Wang, X. Zhao, D. Pei, R. Bush, D. Massey, A. Mankin, S. F. Wu, and L. Zhang, “Observation and analysis of BGP behavior under stress,” in *Proc. 2nd ACM SIGCOMM Workshop on Internet Measurement*, New York, NY, USA, 2002, pp. 183–195.

[52] J. Park, D. Jen, M. Lad, S. Amante, D. McPherson, and L. Zhang, “Investigating occurrence of duplicate updates in BGP announcements,” in *Proc. Passive and Active Measurement*, Zurich, Switzerland, April 2010, pp. 11–20.

[53] V. I. Levenshtein, “Binary codes capable of correcting deletions, insertions and reversals,” in *Soviet Physics Doklady*, Technical Report 8, 1966, pp. 707–710.

[54] R. A. Wagner and M. J. Fisher, “The string-to-string correction problem,” *Journal of the ACM*, vol. 21, no. 1, pp. 168–173, January 1974.

[55] C. Labovitz, R. Wattenhofer, S. Venkatachary, and A. Ahuja, “The impact of Internet policy and topology on delayed routing convergence,” in *Proc. IEEE INFOCOM*, Anchorage, Alaska, April 2001, pp. 537–546.

[56] D. Hunyadi, “Rapid Miner E-Commerce,” in *Proc. 12th WSEAS International Conference on Automatic Control, Modelling and Simulation*, Catania, Italy, May 2010, pp. 316–321.

[57] G. Holmes, A. Donkin, and I. H. Witten, “WEKA: a machine learning workbench,” in Proc. 2nd Australian and New Zealand Conference on Intelligent Information Systems , Brisbane, Australia, December 1994, pp. 357–361.

[58] S. Haeri, D. Kresic, and Lj. Trajkovic, “Probabilistic verification of BGP convergence,” in *Proc. IEEE International Conference on Network Protocols, ICNP 2011*, Vancouver, BC, Canada, October 2011, pp. 127-128 (students poster session paper).

[59] B. A. Prakash, N. Valler, D. Andersen, M. Faloutsos, and C. Faloutsos, “BGP-lens: patterns and anomalies in Internet routing updates,” in *Proc. ACM SIGKDD*, Paris, France, July 2009, pp. 1315–1324.

[60] J. A. K. Suykens and J. Vandewalle, “Least squares support vector machine classifiers,” *Neural Processing Letters*, vol. 9, no. 3, pp. 293–300, February 1999.

[61] L. R. Rabiner and B. H. Juang, “An introduction to hidden Markov models,” *IEEE ASSP Magazine*, vol. 3, no. 1, pp. 4–16, January 1986.

Appendices

Appendix A.

C# Code for the extraction of attributes

The C# code was used to extract the BGP *update* features such as number of announcements, number of withdrawals, and average edit distance.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace ConsoleApplication1
{
    class pins
    {
        int Count; // Counter of BGP messages in each pins matches 1 minute
              period  of time
        int Count_as; // Counter of BGP messages that have an AS path
              attribute in it to compute Average AS path (Announcement messages
              only)
        int Count_unique_as; // Counter of BGP messages that have a unique
              AS_path attribute in it to compute the Average AS path member
              variable (Announcement messages only)
        DateTime Time; // Time of the pin during the day
        int NumberOfannouncedPrefixes; // Total number of the announced
              prefixes for each 1 minute period of time
        int NumberOfWithdrawnsPrefixes; // Total number of withdrawn prefixes
              for each 1 minute period of time
        int NumberOfAnnouncments; // Total number of BGP update messages that
              announce NLRIs
        int NumberOfWithdrawals; // Total number of BGP update messages that
              withdraw NLRIs
        int NumberOfUpdates; // Total number of BGP update messages that
              announce or withdraw NLRIs


        double AvgAsPath; // Average AS path length of all the packets for
              each 1 minute period of time
        double MaxAsPath; // Maximum AS path length of all the packets for
              each 1 minute period of time
        double MaxUniqueAsPath;// Maximum AS path length of all the packets
              for each 1 minute period of time that has a unique AS path. It is
              the same as MaxAsPath
        double AvgUniqueAsPath; // Average AS path length of all the packets
              for each 1 minute period of time that has a unique AS path
        List<string> Unique_AS_Path = new List<string>(); // List of all unique
              AS paths in 1 minute period of time

        // Calculate duplicates messages
        int DuplicateBGPAnnouncements; // Number of duplicate BGP announcement
              messages
```

```
List<bgp_updates> PinsBGPUpdates = new List<bgp_updates>();
int DuplicateBGPWithdrawls; // Number of duplicate BGP withdrawal
                              messages
int NADA;// Number of BGP messages that have new announcements but
            different attributes
int ImplicitWithdrawals; // Number of BGP messages that were announced
      before and are again announced again with different AS path

// Edit Distance
int MaximumAsPathEditDistnace; // Maximum edit distance for the AS
      path attribute
int AverageAsPathEditDistnace; // Average edit distance for the AS
      path attribute
int MinimumAsPathEditDistnace; // Minimum edit distance for the AS
      path attribute


// Origin
int NumberOfIGP; // Number of IGP BGP packets for each 1 minute period
      of time
int NumberOfEGP; // Number of EGP BGP packets for each 1 minute period
      of time
int NumberOfIncomplete; // Number of incomplete BGP packets for each 1
      minute period of time


// Open + Keepalive + Notification
int NumberOfOPENMessages; // Number of open messages for each 1 minute
      period of time
int NumberOfKeepAliveMessages; // Number of keepalive messages for
      each 1 minute period of time
int NumberOfUPDATEMessages; // Number of update messages for
      each 1 minute period of time. It is also used as counter for number
      of messages
int NumberOfNOTIFICATIONMessages; Number of notification messages for
      each 1 minute period of time

// Average size
int AvgSize; // Average packet size in bytes

// Properties
public int count
{
    get { return Count; }
    set { Count = value; }
}

public int AVGSize
{
    get { return AvgSize; }
    set { AvgSize = value; }
}
public int numberOfOPENMessages
{
    get { return NumberOfOPENMessages; }
    set { NumberOfOPENMessages = value; }
}
public int numberOfKeepAliveMessages
{
    get { return NumberOfKeepAliveMessages; }
    set { NumberOfKeepAliveMessages = value; }
}
public int numberOfUPDATEMessages
{
    get { return NumberOfUPDATEMessages; }
    set { NumberOfUPDATEMessages = value; }
}
public int numberOfNOTIFICATIONMessages
{
    get { return NumberOfNOTIFICATIONMessages; }
    set { NumberOfNOTIFICATIONMessages = value; }
}

public int numberOfIGP
{
    get { return NumberOfIGP; }
    set { NumberOfIGP = value; }
}
public int numberOfEGP
{
    get { return NumberOfEGP; }
    set { NumberOfEGP = value; }
}
public int numberOfIncomplete
{
    get { return NumberOfIncomplete; }
    set { NumberOfIncomplete = value; }
}

public int minimumAsPathEditDistance
{
    get { return MinimumAsPathEditDistnace; }
    set { MinimumAsPathEditDistnace = value; }
}
public int averageAsPathEditDistnace
{
    get { return AverageAsPathEditDistnace; }
    set { AverageAsPathEditDistnace = value; }
}

public int maximumAsPathEditDistnace
{
    get { return MaximumAsPathEditDistnace; }
    set { MaximumAsPathEditDistnace = value; }
}

public int duplicateBGPWithdrawls
{
    get { return DuplicateBGPWithdrawls; }
    set { DuplicateBGPWithdrawls = value; }
}
public int nADA
```

```
{
    get { return NADA; }
    set { NADA = value; }
}
public int implicitWithdrawals
{
    get { return ImplicitWithdrawals; }
    set { ImplicitWithdrawals = value; }
}


public List<bgp_updates> pinsBGPUpdates
{
    get { return PinsBGPUpdates; }
    set { PinsBGPUpdates = value; }
}


public int duplicateBGPAnnouncements
{
    get { return DuplicateBGPAnnouncements; }
    set { DuplicateBGPAnnouncements = value; }
}


public int count_as
{
    get { return Count_as; }
    set { Count_as = value; }
}
public int count_unique_as
{
    get { return Count_unique_as; }
    set { Count_unique_as = value; }
}
public DateTime time
{
    get { return Time; }
    set { Time = value; }
}

public int NumberOfAnnouncedPrefixes
{
    get { return NumberOfannouncedPrefixes; }
    set { NumberOfannouncedPrefixes = value; }
}
public int NumberOfwithdrawnsPrefixes
{
    get { return NumberOfWithdrawnsPrefixes; }
    set { NumberOfWithdrawnsPrefixes = value; }
}
public int NumberofAnnouncments
{
    get { return NumberOfAnnouncments; }
    set { NumberOfAnnouncments = value; }
}
```

```
        public int NumberofWithdrawals
        {
            get { return NumberOfWithdrawals; }
            set { NumberOfWithdrawals = value; }
        }
        public int NumberofUpdates
        {
            get { return NumberOfUpdates; }
            set { NumberOfUpdates = value; }
        }

        public double AvgASPath
        {
            get { return AvgAsPath; }
            set { AvgAsPath = value; }
        }
        public double MaxASPath
        {
            get { return MaxAsPath; }
            set { MaxAsPath = value; }
        }
        public double maxUniqueASPath
        {
            get { return MaxUniqueAsPath; }
            set { MaxUniqueAsPath = value; }
        }
        public double AvgUniqueASPath
        {
            get { return AvgUniqueAsPath; }
            set { AvgUniqueAsPath = value; }
        }

        public List<string> unique_AS_Path
        {
            get { return Unique_AS_Path; }
            set { Unique_AS_Path = value; }
        }
}
```

Appendix B.

C# Code for selecting the BGP attributes

This section contains the C# code used to parse ASCII files and extract desired attributes from BCNET BGP data.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.IO;
using System.Text.RegularExpressions;


namespace ConsoleApplication1
{
    class Program // The main class that contains static main methods
    {
        static void Main(string[] args) // The argument list that the user might
              provide to the main
        {

            List<pins> PINS = new List<pins>();
            List<bgp_updates> BGP_UPDATES = new List<bgp_updates>();

            {
                List<string> Bgp_Messages_text = new List<string>(); // This block
                     is to get rid ofBgp messages text variable
               // Main loop
                     // Parse the input file to bgp_update object so all the BGP
                     update messages will be extracted

                StreamReader streamReader = new StreamReader(args[0]);
                string text = "";
                String line;
                bool ParsingFlag = false;
                bool ProcessingFlag = false;

                // Parsing the loop
                while ((line = streamReader.ReadLine()) != null)
                {

                    if (line.Contains("Time") == true || ParsingFlag == true)
                    {
                        // Termination of the loop
                        if (line == "")
                        {
                            ParsingFlag = false; // Set the parsing flag to OFF
                            ProcessingFlag = true; // Set the processing flag to
                                   ON
                        }
                        else //parsing
                        {
                            ParsingFlag = true; // Set the parsing flag to ON
                            text += line + "\n";
                        }

                    }

                    if (ProcessingFlag == true)
                    {
                        // Ignore Open and Keepalive messages
                        if (text.Contains("Parameter") == false &&
                           text.Contains("Keepalive")==false &&
                           text.Contains("Open")==false &&
                           text.Contains("State")==false)
                        {

                            bgp_updates item = new bgp_updates();
                           string[] temp_array = text.Split(new string[] { "\n" },
                           StringSplitOptions.None);


                        // Get the time
                            temp_array[0] = temp_array[0].Trim();
                            int Years = (int.Parse((temp_array[0].Split(' ')
                                   [1])));
                           int Months = (int.Parse((temp array [0].Split (' ')
                                   [2])));
                           int Days = (int.Parse((temp_array[0].Split(' ')
                                   [3])));

                           int Hours = (int.Parse((temp_array[0].Split(' ')
                                   [4].Split(':')[0])));
                           int Minutes = (int.Parse((temp_array[0].Split(' ')
                                   [4].Split(':')[1])));
                           int Seconds = (int.Parse((temp_array[0].Split(' ')
                                           [4].Split(':')[2])));
                           DateTime date1 = new DateTime(Years, Months, Days,
                                   Hours, Minutes, Seconds);
                            item.date = date1;


                         item.sIZE = 0;
                          foreach (string temp in temp_array)
                          {
                           if (temp.Contains("AS_PATH")
                            item.as_path = temp.Split (':')[1].Trim();


                           if (temp.Contains("ORIGIN")
                            item.origin = temp.Split (':')[1].Trim();


                           if (temp.Contains("PACKET TYPE")
                            item.type = temp.Split (':')[1].Trim();
                           if (temp.Contains("FROM")
                            item.from = temp.Split (':')[1].Trim();


                           if (temp.Contains("TO"))
                           item.to = temp.Split (':')[1].Trim();


                           if (temp.Contains("NEXT_HOP"))
                           item.Next_Hop = temp.Split (':')[1].Trim();

                          if (temp.Contains("ANNOUNCED"))
                           item.announced.Add((temp.Split (':')[1].
                            Trim()).Split('/')[0]);


                          if (temp.Contains("WITHDRAWN"))
                           item.withdrawn.Add((temp.Split (':')[1].
                            Trim()).Split('/')[0]);


                       if (temp.Contains("ORIGIN"))
                           item.origin.Add((temp.Split (':')[1].
                            Trim()).Split('/')[0]);


                        if (temp.Contains("COMMUNITIES"))
                           {
                            string[] stringSeparators = new string[] {
                              "COMMUNITIES:"};
                           item.COMMUNITY_ATTRIBUTTES = temp.Split(
                             stringSeparators, StringSplitOptions.
                             None)[1].Trim() ;
                               }


                   //for getting KEEPALIVE + UPDATE+ OPEN+ NOTIFICATION
                    if (temp.Contains("BGP"))
                    {
                     string[] stringSeparators = new string[] {"TYPE:"};
                     item.type = temp.Split(stringSeparators,
                      StringSplitOptions.None)[1].Trim();
                        }

                            item.size += temp.Length;

                            }//end of for loop

                   BGP_UPDATES.Add(item);
                   Console.WriteLine("1: "+"BGP Update msg date" + item.date +
                    "."+BGP_UPDATES.Count +" Have been parsed.");
                    }// end of if statement

                        text = "";//reset next BGP message
                        ProcessingFlag = false; // Set the processing flag to OFF
                    }
```

```
    }
    //continue
    streamReader.Close();

}// Get rid of the Bgp_Messages_text

     int CounterOfParsedMessages = 0 ;
     bool flag_doitonce = true;// Enable the searching for one count
         of attributes

// Big processing loop to compute attributes
foreach (bgp_updates x in BGP_UPDATES)
{
    pins item = new pins();//Will be added later to PINS[]

    item.time = x.date;
    // First minute
    if(x.date.Second==59 && x.date.Hour==22 && x.date.Minute==12)
        item.time = x.date;
    if (PINS.Count == 0)
    {
        PINS.Add(item);
        // Any new attributes may be added here:
        if (x.Announced.Count != 0)
        {
            PINS[0].NumberofAnnouncments += 1;
            PINS[0].NumberOfAnnouncedPrefixes += x.Announced.Count;
            PINS[0].NumberofUpdates += 1;
        }
        if (x.WITHDRAWN.Count != 0)
        {
            PINS[0].NumberofWithdrawals += 1;
            PINS[0].NumberOfwithdrawnsPrefixes += x.WITHDRAWn.Count;
            PINS[0].NumberofUpdates += 1;
        }

        // AS PATH
        if (x.as_path != null)
        {
            PINS[0].AvgASPath = x.as_path.Split(' ').Length;
            PINS[0].count_as = 1;
            PINS[0].MaxASPath = x.as_path.Split(' ').Length; ;
            PINS[0].AvgUniqueASPath = x.as_path.Split(' ').Length;
            PINS[0].maxUniqueASPath = x.as_path.Split(' ').Length;
            // PINS[0].implicitWithdrawals = 1;// will be solved later
        }
        else
        {
            PINS[0].AvgASPath = 0;
            PINS[0].MaxASPath = 0;
            PINS[0].AvgUniqueASPath = 0;
            PINS[0].maxUniqueASPath = 0;
        }

        // ORIGIN
        if (x.origin == "EGP")
            PINS[0].numberOfEGP = 1;
        else if (x.origin == "Incomplete")
            PINS[0].numberOfIncomplete = 1;
        else
            PINS[0].numberOfIGP = 1;

        // Type
        if (x.type == "UPDATE")
            PINS[0].numberOfUPDATEMessages = 1;
        else if (x.type == "KEEPALIVE")
            PINS[0].numberOfKeepAliveMessages = 1;
        else if (x.type == "NOTIFICATION")
            PINS[0].numberOfNOTIFICATIONMessages = 1;
        else if (x.type == "OPEN")
            PINS[0].numberOfOPENMessages = 1;

        //Size
        PINS[0].AVGSize = x.sIZE;


        // BGP Announcement Types
        // nothing!
        PINS[0].pinsBGPUpdates.Add(x);

        PINS[0].count = 1;

    }
    // Other minutes
    else if (PINS[PINS.Count - 1].time.Hour == item.time.Hour &&
         PINS[PINS.Count - 1].time.Minute == item.time.Minute)
    { // Any new attributes may be added here

        if (x.Announced.Count != 0)
        {
            PINS[PINS.Count - 1].NumberofAnnouncments += 1;
  PINS[PINS.Count - 1].NumberOfAnnouncedPrefixes += x.Announced.Count;
            PINS[PINS.Count - 1].NumberofUpdates += 1;
        }
        if (x.WITHDRAWn.Count != 0)
        {
            PINS[PINS.Count - 1].NumberofWithdrawals += 1;
 PINS[PINS.Count - 1].NumberOfwithdrawnsPrefixes += x.WITHDRAWn.Count;
            PINS[PINS.Count - 1].NumberofUpdates += 1;
        }

        // AS PATH
        if (x.as_path != null)
        {
       PINS[PINS.Count - 1].AvgASPath += x.as_path.Split(' ').Length;
            PINS[PINS.Count - 1].count_as += 1;
     if (x.as_path.Split(' ').Length > PINS[PINS.Count - 1].MaxASPath)
        PINS[PINS.Count - 1].MaxASPath = x.as_path.Split(' ').Length;
        }
        // Unique AS PATH
        if (x.as_path != null)
        {
            // Solve for the first item
            if (PINS[PINS.Count - 1].unique_AS_Path.Count == 0)
            {
                PINS[PINS.Count - 1].unique_AS_Path.Add(x.as_path);
                PINS[PINS.Count - 1].count_unique_as += 1;
            }
            else
         // For other items
         { // Check if the AS PATH is not there
 if (PINS[PINS.Count 1].unique_AS_Path.Contains(x.as_path) == false)
                {
                   PINS[PINS.Count - 1].unique_AS_Path.Add(x.as_path);
                    PINS[PINS.Count - 1].count_unique_as += 1;
                }

        // Maximum unique AS path
if (PINS[PINS.Count - 1].unique_AS_Path[PINS[PINS.Count -
1].unique_AS_Path.Count - 1].Split(' ').Length > PINS[PINS.Count -
1].maxUniqueASPath)
PINS[PINS.Count - 1].maxUniqueASPath = x.as_path.Split(' ').Length;

            }

        }
      // Duplicate BGP packets
        PINS[PINS.Count - 1].pinsBGPUpdates.Add(x);

      // ORIGIN
        if (x.origin == "EGP")
            PINS[PINS.Count - 1].numberOfEGP += 1;
        else if (x.origin == "INCOMPLETE")
            PINS[PINS.Count - 1].numberOfIncomplete += 1;
        else
            PINS[PINS.Count - 1].numberOfIGP += 1;
        // Extract the TYPE of duplicate BGP packets
        if (x.type == "UPDATE")
            PINS[PINS.Count - 1].numberOfUPDATEMessages += 1;
        else if (x.type == "KEEPALIVE")
            PINS[PINS.Count - 1].numberOfKeepAliveMessages += 1;
        else if (x.type == "NOTIFICATION")
            PINS[PINS.Count - 1].numberOfNOTIFICATIONMessages += 1;
        else if (x.type == "OPEN")
            PINS[PINS.Count - 1].numberOfOPENMessages += 1;

        // Size
        PINS[PINS.Count - 1].AVGSize += x.sIZE;
        PINS[PINS.Count - 1].count += 1;


  }
else if (PINS[0].time.Hour == item.time.Hour &&PINS[0].time.Minute
== item.time.Minute) // For those BGP messages that come late and
belong to the first attribute

  {// Any new attribute may be added here:
```

```
if (x.Announced.Count != 0)
{
    PINS[0].NumberofAnnouncments += 1;
    PINS[0].NumberOfAnnouncedPrefixes += x.Announced.Count;
    PINS[0].NumberofUpdates += 1;
}
if (x.WITHDRAWN.Count != 0)
{
    PINS[0].NumberofWithdrawals += 1;
    PINS[0].NumberOfwithdrawnsPrefixes += x.WITHDRAWn.Count;
    PINS[0].NumberofUpdates += 1;
}

// AS PATH
if (x.as_path != null)
{
    PINS[0].AvgASPath += x.as_path.Split(' ').Length;
    PINS[0].count_as += 1;
    if (x.as_path.Split(' ').Length > PINS[0].MaxASPath)
        PINS[0].MaxASPath = x.as_path.Split(' ').Length;
}
// Unique AS PATH
if (x.as_path != null)
{
    // Solve for the first item
    if (PINS[0].unique_AS_Path.Count == 0)
    {
        PINS[0].unique_AS_Path.Add(x.as_path);
        PINS[0].count_unique_as += 1;
    }
    else// For other items
    {   // Check if the AS PATH is not there
        if (PINS[0].unique_AS_Path.Contains(x.as_path) ==
       false)
        {
            PINS[0].unique_AS_Path.Add(x.as_path);
            PINS[0].count_unique_as += 1;
        }

        // MaxUniqueAsPath
        if
       (PINS[0].unique_AS_Path[PINS[0].unique_AS_Path.Count -
       1].Split(' ').Length > PINS[0].maxUniqueASPath)
      PINS[0].maxUniqueASPath = x.as_path.Split(' ').Length;

    }
}

// ORIGIN
if (x.origin == "EGP")
    PINS[0].numberOfEGP += 1;
else if (x.origin == "Incomplete")
    PINS[0].numberOfIncomplete += 1;
else
    PINS[0].numberOfIGP += 1;
    // TYPE of ORIGIN attribute
    if (x.type == "UPDATE")
        PINS[0].numberOfUPDATEMessages = 1;
    else if (x.type == "KEEPALIVE")
        PINS[0].numberOfKeepAliveMessages += 1;
    else if (x.type == "NOTIFICATION")
        PINS[0].numberOfNOTIFICATIONMessages += 1;
    else if (x.type == "OPEN")
        PINS[0].numberOfOPENMessages += 1;


    // Calculate size of the packet
    PINS[0].AVGSize += x.sIZE;
    PINS[PINS.Count - 1].count += 1;

}
else// Go forward to next pin
{
     // Iniliaze the first pin as pin zero just in case there are
     no other pins
    PINS.Add(item);

    // Move to the last else
    // if (PINS.Count > 2)
    // if (PINS[PINS.Count -1].count == 1) //for those pins which
     have just one packet
   //{
  // flag_doitonce = false;//disable searching for the previous
     pin
      // PINS.Add(item);
      // Any new attributes may be added here:
            if (x.Announced.Count != 0)
            {
               PINS[PINS.Count - 1].NumberofAnnouncments = 1;
               PINS[PINS.Count - 1].NumberOfAnnouncedPrefixes = 1;
                // PINS[PINS.Count - 1].NumberofUpdates = 1;
            }
            if (x.WITHDRAWN.Count != 0)
            {
                PINS[PINS.Count - 1].NumberofWithdrawals = 1;
              PINS[PINS.Count - 1].NumberOfwithdrawnsPrefixes = 1;
                // PINS[PINS.Count - 1].NumberofUpdates = 1;
            }

            // AS PATH
            if (x.as_path != null)
            {

    PINS[PINS.Count - 1].AvgASPath =  x.as_path.Split(' ').Length;
                PINS[PINS.Count - 1].count_as = 1;
                PINS[PINS.Count - 1].count_unique_as = 1;
     // PINS[PINS.Count - 1].implicitWithdrawals = 1;
     // will be solved later
                PINS[PINS.Count - 1].MaxASPath = 1;
                PINS[PINS.Count - 1].AvgUniqueASPath = 1;
```

```
                        PINS[PINS.Count - 1].maxUniqueASPath = 1;
                    }
                    //else
                    //{
                   // PINS[PINS.Count - 1].AvgASPath = x.as_path.Split('
                   ').Length;
                    // PINS[PINS.Count - 1].MaxASPath = 1;
                    // PINS[PINS.Count - 1].AvgUniqueASPath = 1;
                    // PINS[PINS.Count - 1].maxUniqueASPath = 1;
                    //}

                    // ORIGIN
                    if (x.origin == "EGP")
                        PINS[PINS.Count - 1].numberOfEGP = 1;
                    else if (x.origin == "Incomplete")
                        PINS[PINS.Count - 1].numberOfIncomplete = 1;
                    else
                        PINS[PINS.Count - 1].numberOfIGP = 1;

                    // TYPE
                    if (x.type == "UPDATE")
                        PINS[PINS.Count - 1].numberOfUPDATEMessages = 1;
                    else if (x.type == "KEEPALIVE")
                       PINS[PINS.Count - 1].numberOfKeepAliveMessages = 1;
                    else if (x.type == "NOTIFICATION")
                    PINS[PINS.Count - 1].numberOfNOTIFICATIONMessages = 1;
                    else if (x.type == "OPEN")
                        PINS[PINS.Count - 1].numberOfOPENMessages = 1;


                    // Size
                    PINS[PINS.Count - 1].AVGSize = 1;
                    PINS[PINS.Count - 1].count = 1;

                }

        //Logging //"PINS.Count ="+PINS.Count +
        Console.WriteLine("2: "+" x.date =" + x.date + " i=" +
             CounterOfParsedMessages +" have been processed.");
        CounterOfParsedMessages++;

    }

    // Compute Duplicate BGP packets

    int counterOfProcessedMessages = 0;
    foreach (pins pin in PINS)//for all the pins
    {
        for (int i = 0; i < pin.pinsBGPUpdates.Count; i++) //take 1 pin
        {
             // for (int j = pin.pinsBGPUpdates.Count - 1 - i; j <
                    pin.pinsBGPUpdates.Count; j++) //cross
        for (int j =0; j <=i; j++) //cross new
            {
                foreach (string plapla in pin.pinsBGPUpdates[i].Announced)
                    if (pin.pinsBGPUpdates[j].Announced.Contains(plapla))
                    {

            if (pin.pinsBGPUpdates[j].as_path ==
                   pin.pinsBGPUpdates[i].as_path)// Duplicate
                        {
                            pin.duplicateBGPAnnouncements += 1;
                        }
            else if (pin.pinsBGPUpdates[j].as_path !=
                   pin.pinsBGPUpdates[i].as_path) // Implicit Withdrawal
                        {
                            pin.implicitWithdrawals += 1;
                        }

                    }

            // Check for duplicate Withdrawals
                  foreach (string plapla in pin.pinsBGPUpdates[i].WITHDRAWn)
                   if (pin.pinsBGPUpdates[j].WITHDRAWN.Contains(plapla))
                          {
                              pin.duplicateBGPWithdrawls += 1;
                          }
                  }
              }
              CounterOfProcessedMessages++;
              Console.WriteLine("3: "+"Duplicate for pin number: " +
               CounterOfProcessedMessages+" is bieng prcoessed.
              ("+ pin.count_as+" bgp msgs)");
          }

// Calculate the Average AS path length
  for (int i = 0; i < PINS.Count; i++)
      {
         // Fix AS_PATH sum
         // PINS[i].AvgASPath = Math.Round(PINS[i].AvgASPath / PINS[i].count);
              PINS[i].AvgASPath = Math.Ceiling((PINS[i].AvgASPath /
            PINS[i].count_as) );//*100 to make it obvious between the
            AvgUniqueASPath and AvgASPath

          // Fix Unique AS-PATH count + sum
              foreach (string temp in PINS[i].unique_AS_Path)
              {
                  PINS[i].AvgUniqueASPath += temp.Split(' ').Length;
              }
                   PINS[i].AvgUniqueASPath =
                   Math.Ceiling((PINS[i].AvgUniqueASPath /
                   PINS[i].count_unique_as));
              if(PINS[i].count!=0)
                  PINS[i].AVGSize =(int)(Math.Ceiling((PINS[i].AVGSize /
                   PINS[i].count)*1.0));
          }


          // Compute the Max Edit distance
          int NumberOfEditDistanceCalculatedPins = 1 ;
          foreach (pins temp in PINS)
          {
              // Get all the as-path from each packet
              List<string> to_send = new List<string>();
              foreach (bgp_updates x1 in temp.pinsBGPUpdates)
                  if (x1.as_path != null && to_send.Contains(x1.as_path) ==
                   false)
                  {
          // Get all unique ASs so it won't be the same as max as path length
             HashSet<string> items = new HashSet<string>(x1.as_path.Split(' '));
                    // to_send.Add(x1.as_path);
                      to_send.Add(String.Join(" ",items.ToArray()));
                  }


              Console.WriteLine("4: "+"Computing Edit Distance for pin= " +
              counter + ", that contains= " + temp.count_as + " bgp msgs.");
              int[] output = MaxEditDistnace(to_send);
              temp.maximumAsPathEditDistnace = output[0];
              temp.averageAsPathEditDistnace = output[1];
              temp.minimumAsPathEditDistnace = output[2];
              counter++;
          }

// Writing to the stdout

// TextWriter tw = new StreamWriter("date_test_bcnet.txt");
// TextWriter tw2 = new StreamWriter("date_test_bcnet.txt"
     +"_attributeextraction");
      TextWriter streamtwriter1 = new StreamWriter(args[1]);
      TextWriter streamwriter2 = new StreamWriter(args[1]+"_attribute
            selection");
          for (int i = 0; i < PINS.Count; i++)
          {
              string SecondToPrint;
              if (PINS[i].time.Second < 10)
                  SecondToPrint = "0" + PINS[i].time.Second.ToString();
              else
                  SecondToPrint = PINS[i].time.Second.ToString();

              string MinuteToPrint;
              if (PINS[i].time.Minute < 10)
                  MinuteToPrint = "0" + PINS[i].time.Minute.ToString();
              else
                  MinuteToPrint = PINS[i].time.Minute.ToString();

              string HourToPrint;
              if (PINS[i].time.Hour < 10)
                  HourToPrint = "0" + PINS[i].time.Hour.ToString();
              else
                  HourToPrint = PINS[i].time.Hour.ToString();


          string output18 = ((Math.Round(PINS[i].MaxASPath) == 11) ? "1" : "0");
          string output19 = ((Math.Round(PINS[i].MaxASPath) == 12) ? "1" : "0");
          string output20 = ((Math.Round(PINS[i].MaxASPath) == 13) ? "1" : "0");
          string output21 = ((Math.Round(PINS[i].MaxASPath) == 14) ? "1" : "0");
          string output22 = ((Math.Round(PINS[i].MaxASPath) == 15) ? "1" : "0");
```

```
            string output23 = ((Math.Round(PINS[i].MaxASPath) == 16) ? "1" : "0");
            string output24 = ((Math.Round(PINS[i].MaxASPath) == 17) ? "1" : "0");
            string output25 = ((Math.Round(PINS[i].MaxASPath) == 18) ? "1" : "0");
            string output26 = ((Math.Round(PINS[i].MaxASPath) == 19) ? "1" : "0");
            string output27 = ((Math.Round(PINS[i].MaxASPath) == 20) ? "1" : "0");

string output28 = ((Math.Round(PINS[i].maximumAsPathEditDistnace * 1.0) == 7) ?
"1" : "0");
string output29 = ((Math.Round(PINS[i].maximumAsPathEditDistnace * 1.0) == 8) ?
"1" : "0");
string output30 = ((Math.Round(PINS[i].maximumAsPathEditDistnace * 1.0) == 9) ?
"1" : "0");
 string output31 = ((Math.Round(PINS[i].maximumAsPathEditDistnace * 1.0) == 10) ?
"1" : "0");
 string output32 = ((Math.Round(PINS[i].maximumAsPathEditDistnace * 1.0) == 11) ?
"1" : "0");
 string output33 = ((Math.Round(PINS[i].maximumAsPathEditDistnace * 1.0) == 12) ?
"1" : "0");
 string output34 = ((Math.Round(PINS[i].maximumAsPathEditDistnace * 1.0) == 13) ?
"1" : "0");
 string output35 = ((Math.Round(PINS[i].maximumAsPathEditDistnace * 1.0) == 14) ?
"1" : "0");
 string output36 = ((Math.Round(PINS[i].maximumAsPathEditDistnace * 1.0) == 15) ?
"1" : "0");
 string output37 = ((Math.Round(PINS[i].maximumAsPathEditDistnace * 1.0) == 16) ?
"1" : "0");
//string output38 = ((Math.Round(PINS[i].maximumAsPathEditDistnace * 1.0) == 17) ?
"1" : "0"); // gives all zeros


                // Text Format
                Console.WriteLine(
                    HourToPrint + MinuteToPrint + " " +//1
                    HourToPrint +" " +//2
                    MinuteToPrint +" "+ //3
                    SecondToPrint+" "+ //4
                    PINS[i].NumberofAnnouncments + " " +//5
                    PINS[i].NumberofWithdrawals + " " +//6
                            // PINS[i].NumberofUpdates + " " + //7
                    PINS[i].NumberOfAnnouncedPrefixes + " " +//7
                    PINS[i].NumberOfwithdrawnsPrefixes + " " +//8
                    PINS[i].AvgASPath + " " +//9
                    PINS[i].MaxASPath + " " +//10
                    PINS[i].AvgUniqueASPath + " " +//11
                    PINS[i].duplicateBGPAnnouncements + " " + //12
                    PINS[i].implicitWithdrawals + " " +//13
                    PINS[i].duplicateBGPWithdrawls + " " +//14
                    PINS[i].maximumAsPathEditDistnace + " " +//15
                    ((1 / (60.0 / Math.Ceiling(PINS[i].pinsBGPUpdates.Count *
                     1.0))).ToString() + "000").Substring(0, 3) + " " +//16 //4.3
                     //wrong
                     // PINS[i].maximumAsPathEditDistnace + " " + //18
                    PINS[i].averageAsPathEditDistnace + " " +//17


                     output18 +" "+//18
```

```
                                output19 +" "+//19
                                output20 +" "+//20
                                output21 +" "+//21
                                output22 +" "+//22
                                output23 +" "+//23
                                output24 +" "+//24
                                output25 +" "+//25
                                output26 +" "+//26
                                output27 +" "+//27

                                output28 +" "+//28
                                output29 +" "+//29
                                output30 +" "+//30
                                output31 +" "+//31
                                output32 +" "+//32
                                output33 +" "+//33
                                output34 +" "+//34
                                output35 +" "+//35
                                output36 +" "+//36
                                output37 + " " +//37


                                PINS[i].numberOfIGP + " " + //38
                                PINS[i].numberOfEGP + " " + //39
                                PINS[i].numberOfIncomplete +" "+ //40


                                //PINS[i].numberOfUPDATEMessages + " " + //41
                                //PINS[i].numberOfOPENMessages + " " + //42
                                //PINS[i].numberOfNOTIFICATIONMessages + " " + //43
                                //PINS[i].numberOfKeepAliveMessages + " "+//44

                                PINS[i].AVGSize//45 //41

                               ); //17


                        // For matlab
                        tw.WriteLine(
                            HourToPrint + MinuteToPrint + " " //1
                            + HourToPrint + " " +//2
                            MinuteToPrint + " " + //3
                            SecondToPrint + " " +//4
                            PINS[i].NumberofAnnouncments + " " + //5
                            PINS[i].NumberofWithdrawals + " " + //6
                            PINS[i].NumberOfAnnouncedPrefixes + " " + //7
                            PINS[i].NumberOfwithdrawnsPrefixes + " " +//8
                            PINS[i].AvgASPath + " " +//9
                            PINS[i].MaxASPath + " " +//10
                            PINS[i].AvgUniqueASPath + " " +//11
                            PINS[i].duplicateBGPAnnouncements + " " + //12
                            PINS[i].implicitWithdrawals + " " +//13
                            PINS[i].duplicateBGPWithdrawls + " " +//14
                            PINS[i].maximumAsPathEditDistnace + " " +//15
                            ((1 / (60.0 / Math.Ceiling(PINS[i].pinsBGPUpdates.Count *
1.0))).ToString() + "000").Substring(0, 3) + " " +//16 //4.3
```

```
                       PINS[i].averageAsPathEditDistnace +" "+//17
                        PINS[i].numberOfIGP + " " + //38
                        PINS[i].numberOfEGP + " " + //39
                        PINS[i].numberOfIncomplete +" " +//40

                        //PINS[i].numberOfUPDATEMessages + " " + //41
                       //PINS[i].numberOfOPENMessages + " " + //42
                              usually zereos are neglected
                       //PINS[i].numberOfNOTIFICATIONMessages + " " + //43
                              usually zeros are neglected
                       //PINS[i].numberOfKeepAliveMessages + " " +//44
                              usually zeros are neglected

                        PINS[i].AVGSize//45 //41
                       );

                   //For attributes selection
                   if (i <= 800)
                       tw2.Write("-1 ");
                   else
                       tw2.Write("1 ");

                   tw2.WriteLine(
                     "1:" +
                     PINS[i].NumberofAnnouncments + " 2:" + //1
                     PINS[i].NumberofWithdrawals + " 3:" + //2
                     PINS[i].NumberOfAnnouncedPrefixes + " 4:" + //3
                     PINS[i].NumberOfwithdrawnsPrefixes + " 5:" +//4
                     PINS[i].AvgASPath + " 6:" +//5
                     PINS[i].MaxASPath + " 7:" +//6
                     PINS[i].AvgUniqueASPath + " 8:" +//7
                     PINS[i].duplicateBGPAnnouncements + " 9:" + //8
                     PINS[i].implicitWithdrawals + " 10:" +//9
                     PINS[i].duplicateBGPWithdrawls + " 11:" +//10
                     PINS[i].maximumAsPathEditDistnace + " 12:" +//11
                     ((1 / (60.0 / Math.Ceiling(PINS[i].pinsBGPUpdates.Count *
1.0))).ToString()+"000").Substring(0,3) + " 13:" +//12
                     PINS[i].averageAsPathEditDistnace+" 14:"+//13
                    PINS[i].numberOfIGP + " 35:" + //34
                     PINS[i].numberOfEGP + " 36:" + //35
                        PINS[i].numberOfIncomplete + " 37:" + //36


                        //PINS[i].numberOfUPDATEMessages + " 38:" + //37
                        //PINS[i].numberOfOPENMessages + " 39:" + //38
                        //PINS[i].numberOfNOTIFICATIONMessages + " 40:" + //39
                        //PINS[i].numberOfKeepAliveMessages + " 41:" + //40

                        PINS[i].AVGSize//41 //37
                     );
               }


               streamwriter1.Close();
               streamwriter2.Close();
```

```
        }//end of main

/// <summary>
/// This function is used to compute the Maximum Edit distance from a collection
    of edit distances
/// </summary>
/// <parameter name="a">It is a list of edit distances </parameter>
/// <returns></returns>
public static int [] MaxEditDistance (List<string> a)
  {
    int max = 0;
    int min = 1000;
    int sum = 0;

// break AS path to a list of strings
List<string[]> AsPathList = new List<string[] >() ;
foreach (string x in a)
{
   AsPathList.Add(x.Split((' ');
}
for (int i=0; i < a.Count; i++)
{
  // for (int j = a.Count - 1 - i; j < a.Count; j++) // wrong
    for (int j =0; j <= i; j++)
      {
        int current = Editdistance(AsPathlist[i], AsPathList[j]);
        sum += current;
        if (current > max))
           max = current;
        if (current < min && i != j) // Avoid zero distance
           min =  current;
        // if (current = 0 && i != j) // should not have this value
        // Console. Writeline("Real Zero");

        }
   }
   int [] temp = new int [3];
   temp[0] = max;
   temp[1] = (int)(Math.Ceiling((sum * 1.0)/ (a.Count * a.Count)));
   temp[2] = min;

   // return MeshMatrix.Cast<int>().Max() ;
   return temp ;
  }

/// <summary>
/// This function is used to compute the Edit distance between two AS paths
/// </summary>
/// <parameter name="a"> The first AS path</parameter>
/// <parameter name="a"> The second AS path</parameter>
/// <returns>It returns an integer with the value if the edit distance
    </returns>
public static int Editdistance(string[] a, string [] b)
{
  // for all i and j, d[i,j] will hold the Levenshtein distance between
  // the first i characters of s and the first j characters of t;
```

```
 // note that d has (m+1)x(n+1) values
 // declare int d[0...m, 0...n]
 int [,] EditDistanceArray = new int[a.Length + 1, b.Length + 1];
 // for i from 0 to m
// d [i,0] := i // the distance of any first string to an empty second string
// for j from 0 to n
// d [0,j] := j // the distance o f any second string to an empty first string

 for (int i = 0 ; i < a.Length + 1 ; i++)
     EditDistanceArray [i,0] = i ;
 for (int j = 0 ; j < b.Length + 1 ; j++)
     EditDistanceArray [0,j] = j ;

 for (int i = 1 ; i <= a.Length ; i++)
{
  for (int j = 1 ; j <= b.Length ; j++)
   {
     // if (a[i] == b[j] && i == 0 && j == 0)
     // EditDistanceArray [i ,j] = 0 ;
if (a[i - 1] == b [j - 1])
    EditDistanceArray [i,j] = EditDistanceArray [i - 1,j - 1 ] ;
else
   EditDistanceArray [i ,j] = Math.Min(
    EditDistanceArray [i -1,j] + 1,
     Math .Min(
     EditDistanceArray [i,j - 1] + 1 ,
     EditDistanceArray [i - 1,j - 1] + 1)
    );
  }
}

// return d [m,n]
return EditDistanceArray [a.Length,b.Length] ;

       }
    }
  }
```

Appendix C.

MATLAB Code for generating the graphs

This section contains the sample MATLAB code used to create the graphs after the desired BGP update attributes were extracted. It imports the datasets so that results and graphs may be printed.

```
clear;

clc;

close all;

addpath('C:\Users\lally\Desktop\BGP_thesis\C#code');

addpath('../../matlab');


% Import the datasets

updates0x2E20030123_rcc4 = importdata('nov2results.txt');


% Add a breakpoint after the dataset you wished to import has been imported

updates0x2E20030124_rcc4 = importdata('updates.20030124_rcc4.out');

updates0x2E20030125_rcc4 = importdata('updates.20030125_rcc4.out');

updates0x2E20030126_rcc4 = importdata('updates.20030126_rcc4.out');

updates0x2E20030127_rcc4 = importdata('updates.20030127_rcc4.out');


updates0x2E20010916_rcc4 = importdata('updates.20010916_rcc4.out');

updates0x2E20010917_rcc4 = importdata('updates.20010917_rcc4.out');

updates0x2E20010918_rcc4 = importdata('updates.20010918_rcc4.out');

updates0x2E20010919_rcc4 = importdata('updates.20010919_rcc4.out');

updates0x2E20010920_rcc4 = importdata('updates.20010920_rcc4.out');


updates0x2E20010712_rcc4 = importdata('updates.20010712_rcc4.out');

updates0x2E20010713_rcc4 = importdata('updates.20010713_rcc4.out');

updates0x2E20010714_rcc4 = importdata('updates.20010714_rcc4.out');

updates0x2E20010715_rcc4 = importdata('updates.20010715_rcc4.out');

updates0x2E20010716_rcc4 = importdata('updates.20010716_rcc4.out');


updates0x2E20010717_rcc4 = importdata('updates.20010717_rcc4.out');

updates0x2E20010718_rcc4 = importdata('updates.20010718_rcc4.out');
```

```
updates0x2E20010719_rcc4 = importdata('updates.20010719_rcc4.out');
updates0x2E20010720_rcc4 = importdata('updates.20010720_rcc4.out');
updates0x2E20010721_rcc4 = importdata('updates.20010721_rcc4.out');
```