ANALYSIS OF TRAFFIC DATA FROM A HYBRID SATELLITE-TERRESTRIAL NETWORK

by

Savio Lau

B.A.Sc., Simon Fraser University, 2003

THESIS SUBMITTED IN PARTIAL FULFILLMENT
OF THE REQUIREMENTS FOR THE DEGREE OF

MASTER OF APPLIED SCIENCE

In the
School
of
Engineering Science

SIMON FRASER UNIVERSITY

2007

APPROVAL

Name:	Savio Lau
Degree:	Master of Applied Science
Title of Thesis:	Analysis of Traffic Data from a Hybrid Satellite-Terrestrial Network
Examining Committee:	Dr. Carlo Menon Assistant Professor of School of Engineering Science Chair
	Dr. Ljiljana Trajković Senior Supervisor Professor of School of Engineering Science
	Dr. William Gruver Supervisor Professor Emeritus of School of Engineering Science
	Dr. Stephen Hardy Internal Examiner Professor of School of Engineering Science
Date Approved:	July 31, 2007

Abstract

Satellite data networks provide broadband access for areas not served by traditional broadband technologies. In this thesis, we describe a collection of traffic data (billing records and *tcpdump* traces) from a satellite Internet service provider in China. We use the billing records to investigate the downloaded and uploaded traffic volume and the aggregate user behavior. We examine daily and weekly cycles and effects of holidays on traffic patterns. We also employ cluster analysis methods to classify the users according to their traffic. Analysis of the *tcpdump* traces indicates that transmission control protocol (TCP) accounts for the majority of data transfers. The analysis also includes the detection of anomalies such as invalid TCP flag combinations, port scans, and anomalies in traffic volume.

Keywords: Satellite-terrestrial networks, TCP, traffic measurements, cluster analysis, anomaly detection

Acknowledgments

First and foremost, I would like to thank my senior supervisor Dr. Ljiljana Trajković for her guidance, knowledge, and support throughout my graduate studies. I would also like to thank Dr. William Gruver and Dr. Stephen Hardy for their insights and for serving on my supervisory committee.

I am also very thankful for the wonderful colleagues in the Communications Networks Laboratory. Special thanks go to Modupe Omeuti and Renju Narayanan, whose encouragements and assistance made the completion of this thesis possible. Furthermore, I would like to acknowledge Kenny Shao, whose initial work lead to this research project.

I am also grateful for the assistance provided by Mr. Yongxin Shi and Mr. Rui Huang, the managers of ChinaSat, who made the collection of the billing records and *tcpdump* traces possible.

Lastly, I would like to thank my family and friends, who were always there to support me regardless of the circumstances. My sincere thanks go to Christopher Huang, Michael King, Benjamin Ong, and Yvonne Wai, who gave me support and feedback throughout my studies.

Contents

List of Figures

List of Tables

Glossary

ACK	TCP acknowledgement
ARIMA	Autoregressive integrative moving average
ARP	Address resolution protocol
BER	Bit error rate
BDP	Bandwidth-delay product
CDF	Cumulative distribution function
CPCC	Cophenetic correlation coefficient
CWND	TCP congestion window
DDoS	Distributed denail of service
DF	IP "Do not fragment" flag
DHCP	Dynamic host configuration protocol
DoS	Denial of service
DWT	Discrete wavelet transform
FTP	File transfer protocol
GEO	Geosynchronous earth orbit
HTTP	Hypertext transfer protocol
ICMP	Internet control message protocol

IHL	IP header length
IRC	Internet relay chat protocol
IP	Internet protocol
LAN	Local area network
LEO	Low earth orbit
LSB	Least significant byte
MMSE	Minimum mean-squared error
MSB	Most significant byte
MSE	Mean-squared error
MEO	Medium earth orbit
MTU	Maximum transmission unit
NAT	Network address translation
NETBEUI	NETBIOS extended user interface
NOC	Network operations center
NOP	TCP no operation option
NGEO	Non-geosynchronous earth orbit
OS	Operating system
POP3	Post office protocol version 3
PEP	Performance enhancing proxies
PSH	TCP PSH flag
RARP	Reverse address resolution protocol
RIP	Routing information protocol
RST	TCP RST flag
RTT	Round-trip time

SACK	TCP Selective acknowledgment
SACKOK	TCP SACK permitted option
SC	Silhouette coefficient
SMTP	Simple mail transfer protocol
SSL	Secure socket layer
SYN	TCP SYN flag
TCP	Transmission control protocol
ToS	IP type of service field.
TTL	Time to live
UDP	User datagram protocol
UTC	Coordinated universal time
WSCALE	TCP sliding window scale option
WWW	World wide web

Chapter 1

Introduction

Continuous increase in demand for broadband Internet access has resulted in the increased volume of data traffic and development of new protocols and access technologies. Measurements and analysis of genuine network traffic traces have been used to understand traffic dynamics, characterize traffic, develop new traffic models, and ultimately evaluate network performance.

1.1 Motivations

During the past decade, various traces of wired and wireless terrestrial Internet traffic data have been collected and characterized. Traffic traces collected from university campuses and research institutions have been made publicly available [1]. However, few traces from commercial satellite networks have been made available to the research community. Hence, analysis of traffic data from deployed networks such as ChinaSat is important. Such analysis may provide new insights as technology and usage patterns change.

1.2 Related work

Analysis of traffic traces have dealt with identifying characteristics of TCP connections [2] and network traffic patterns [3]–[7]. It has been found that the Internet traffic is bursty and long-range dependent. Analysis of users from deployed networks have found that a small number of users in a network contributes the majority of traffic. Furthermore, the development of new access technologies have resulted in methods to improve characteristics of TCP over lossy links. Many enhancements have been proposed to improve TCP performance over high speed and wireless networks.

Traffic has also been modelled using statistical distributions [8], [9] and cluster analysis [10]. Due to the long-range dependent and non-stationarity nature of Internet traffic, most statistical traffic models do not fully capture the traffic dynamics.

The architecture and performance enhancement of the DirecPC network was alerady described in [11]. By using a split-connection architecture, the performance of TCP was greatly improved. The performance of hypertext transfer protocol (HTTP) connections in the DirecPC environment has also been evaluated [12], [13]. It was found that performance of short-lived connections such as HTTP greatly depends on the enhancements made to the TCP slow-start algorithm due to the long delays present in satellite links.

Detailed statistical analysis of the DirecPC traffic data was previously reported [14], where the TCP connection inter-arrival time and the number of downloaded bytes were modelled. The inter-arrival time is best modeled by the Weibull distribution while the number of downloaded bytes is best modeled by the lognormal distribution. Based on the IP addresses, the distribution of visited websites is best modeled by the discrete Gaussian exponential (DGX) distribution. Furthermore, the ChinaSat traffic is determined to be self-similar and non-stationary, where measure of self-similarity (the Hurst parameter) differs depending on traffic load.

Traffic was also predicted using the autoregressive integrative moving average (ARIMA) model and a wavelet / auto-regressive model. ARIMA model was able to predict the uploaded traffic but not the downloaded traffic due to the traffic dynamics. The wavelet + autoregressive model was found to outperform the ARIMA model.

In recent years, researchers have focused on the classification of network anomalies [15]–[17]. Statistical methods [18], wavelets [19], and principal component analysis [20] have been employed to detect network anomalies. These methods determine network anomalies by first defining the normal network behavior. Using heuristics, deviation of traffic patterns of one or more connections compared to the normal network behavior are classified as anomalies.

1.3 Contributions

In this thesis, we analyze patterns and statistical properties of two sets of traffic data (billing records, *tcpdump* [21] traces) collected from the DirecPC network. The network is a hybrid satellite-terrestrial network operated by ChinaSat, a commercial satellite Internet service provider located in China.

Using the billing records, we investigate the daily and weekly traffic patterns and the effect of holidays on traffic [22]. We also categorize the network users by employing k-means and hierarchical clustering.

From the *tcpdump* traces, we analyze the general characteristics of the traffic data

[22]. As a measure of network optimization, we examine the Transmission Control Protocol (TCP) [23] options to identify recommended TCP extensions used in the ChinaSat network. To confirm our observations regarding TCP extensions, we employ operating system (OS) fingerprinting techniques to identify TCP implementations. We also investigate traffic anomalies such as invalid TCP flag combinations, port scans, and anomalies in traffic volume [22]. Lastly, we developed the C program *pcapread* to parse *tcpdump* traces.

1.4 Organization

The remaining of this thesis is organized as follows. In Chapter 2, we describe the ChinaSat network architecture and the use of TCP in satellite networks. In Chapter 3, we discuss the mathematical tools employed for our analysis. In Chapter 4, we present the analysis of the billing records, observed traffic patterns, and the user classification using clustering techniques. Analysis of *tcpdump* traces and detection of data traffic anomalies are given in Chapter 5. We conclude with Chapter 6.

Chapter 2

ChinaSat: network architecture and TCP

2.1 DirecPC system

DirecPC is an asymmetric geosynchronous satellite network deployed by Hughes Network Systems. It provides television and data services including: DirecTV (a satellite television service), DirecPC (a unidirectional satellite data service), and DirecWay [24] (a new bidirectional satellite data service intended to replace DirecPC). Turbo Internet is the Internet access component of DirecPC. It provides broadband access through a satellite downlink and a return path through a terrestrial dial-up modem. The downlink path has an advertised rate of 400 kb/s while the uplink path is limited to 33.6 kb/s.

ChinaSat provides Internet access through the DirecPC to individual users, businesses, and over 200 Internet cafés across provinces in China. DirecPC employs TCP splitting with spoofing to improve network performance, which is described in Section 2.3.5. A user's request to browse a website is not sent directly to the destination. Instead, the DirecPC software installed in a satellite user's system adds an Internet Protocol (IP) [25] "tunneling header" to the request IP datagram, redirecting the datagram to the network operations center (NOC). At the NOC, the tunneling header is removed and the request is forwarded to the website using a high-speed terrestrial link. The NOC receives the reply from the website and forwards it to the user via the DirecPC satellite link. Data paths of the DirecPC system are shown in Figure 2.1. IP headers and the tunneling header at the user and the website hosts are shown in Figure 2.2. Each box indicates an IP source/destination pair. At the client (satellite user), the <Satellite IP, Destination IP> datagram is redirected to NOC by embedding the datagram into the tunneling header <Dial-up IP, NOC IP>.

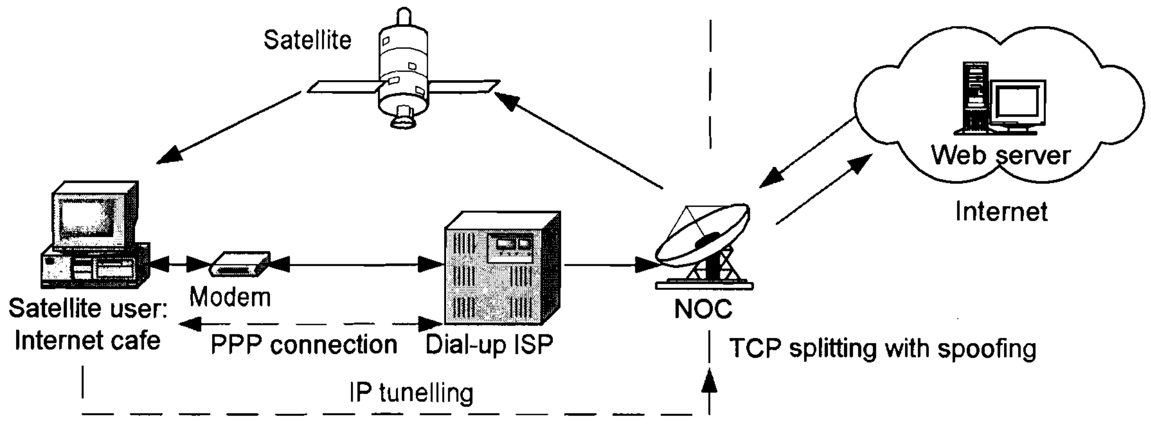

Figure 2.1. Data paths in the DirecPC system.

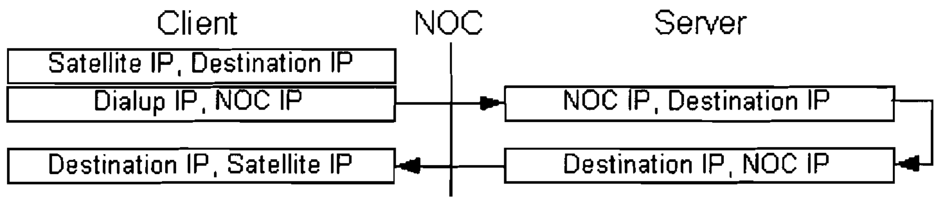

Figure 2.2. IP headers used in the DirecPC system.

2.2 Characteristics of satellite networks

Satellite networks such as the DirecPC broadcast information over a large geographical area and provide the last mile access for remote sites, aircrafts, and ships. Satellites can be classified into two categories: geostationary earth orbit (GEO) and non-GEO (NGEO). GEO satellites orbit at an altitude of ~36,000 km [26], [27]. NGEO satellites can be further classified into low earth orbit (LEO) satellites and medium earth orbit (MEO) satellites. LEO satellites orbit at altitudes 2,000 km or less. MEO satellites orbit between 2,000 and 12,000 km above the earth's surface. Although LEO and MEO satellites have lower propagation delay due to their shorter distance to the earth's surface, GEO satellites have a 24-hour rotation period and appear stationary from the earth's surface. They also have a large footprint, which eliminates the need of tracking equipment for satellite receivers. However, GEO satellites have four undesirable characteristics that degrade TCP performance in satellite links: long propagation delay, large bandwidth-delay product (BDP), high bit error rates (BERs), and path asymmetry.

2.2.1 Long propagation delay

In satellite networks, propagation delay is the dominant component of the total end-to-end delay. GEO links introduce $\sim$250 ms delay for signals travelling between two earth stations via a satellite link [27]. NGEO links have delays from 1 ms to 100 ms.

2.2.2 Large bandwidth-delay product

In addition to the long propagation delay, satellite links also possess large bandwidth. Large bandwidth and long propagation delay result in a large BDP value. BDP indicates the amount of data required to be in transit (unacknowledged) in order to maximize the transfer rate between two connection endpoints. It is defined as

$$\mathrm{BDP} = \mathrm{RTT} \times \mathrm{C}, \tag{2.1}$$

where RTT is the round-trip time delay (seconds) and C is the bandwidth of the satellite link (bits per second). A small TCP window prevents a connection from sending the maximum amount of unacknowledged data specified by BDP.

2.2.3 High bit error rates

Satellite links exhibit higher BERs compared to terrestrial networks, usually in the order of 10^{-6} [28]. BERs in satellite links could degrade to 10^{-3} or 10^{-2} because of propagation losses, noise, and interference. High BERs have a significant impact on networks with long propagation delays. The commonly deployed TCP NewReno may only correct a single missing segment per round-trip time by employing its fast retransmit and fast recovery algorithms. Additional missing segments will cause TCP to enter the slow-start phase, resulting in reduced throughput [27], [29].

2.2.4 Path asymmetry

Satellite networks often employ different bandwidths for uplink and downlink paths. Similar to cellular networks, two-way satellite networks employ a broadcast downlink path and a multiple-access uplink path [27]. Furthermore, the amount of bandwidth may be different due to design and cost considerations. For hybrid satellite-terrestrial networks such as the DirecPC, the uplink path from a user employs terrestrial modems that is limited to 33.6 kb/s while the downlink path from a satellite to a user is 400 kb/s or higher.

2.3 TCP extensions for satellite environments

TCP, the most common Internet transport layer protocol [23], was originally designed for terrestrial wired networks. TCP does not perform well in satellite networks because they exhibit long propagation delay, large bandwidth-delay product, high bit error rates, and path asymmetry, as described in Section 2.2.

Various TCP modifications have improved TCP performance in satellite networks [12], [27], [30]–[33]. The proposed mechanisms to enhance TCP for satellite networks [27] include: the option of increasing the initial TCP congestion window [34], TCP sliding window scale option [29], selective acknowledgement (SACK) option [35], and sending path maximum transmission unit (MTU) discovery [36]. Although recommended, only a small number of these extensions are widely deployed. Performance enhancing proxies (PEPs) [31] have also been successfully deployed to improve TCP performance in satellite networks.

2.3.1 Increasing initial TCP congestion window

TCP avoids transmitting a large burst of data traffic by employing the slow-start algorithm. To begin, the congestion window (*cwnd*) is set to one or two segments depending on the implementation. Subsequently, each received acknowledgement (ACK) during slow-start increases *cwnd* by one segment. In networks with long propagation delays, the sending rate increases slowly because of the delay and the low number of received ACKs. The initial *cwnd* size of $\sim$4 kB (4 segments) has been recommended to increase the number of received ACKs at the beginning of slow-start [12], [35].

2.3.2 TCP sliding window scale option

This extension expands the default TCP window size from 16 bits to 32 bits. The scale factor is included in an 8-bit field named WSCALE of the TCP SYN segment. This field value may be different for each direction. A TCP implementation using the scale factor will right-shift the sliding window by the value of the scale factor [29].

2.3.3 Selective acknowledgment (SACK) option

Fast retransmit and fast recovery algorithms enable TCP NewReno to recover at most one lost segment per round trip time (RTT). However, in environments with high BERs and long propagation delays, additional lost segment(s) cause retransmission time-out(s). Subsequent retransmissions resume from the slow-start phase, which has inferior performance because of a small initial *cwnd.* SACK allows a TCP receiver to explicitly acknowledge the segments that have been received. When a sender identifies lost segment(s) through SACK, it can retransmit earlier and avoid the performance penalty associated with retransmission time-outs and the slow-start algorithm. SACK significantly improves TCP throughput in lossy environments when RTT is large [12]. In TCP implementations, the SACK option is enabled in the TCP SYN packet using TCP option 4 (SACKOK). A TCP receiver identifies missing packets using TCP option 5 (SACK).

2.3.4 Path maximum transmission unit (MTU) discovery

This extension is used to determine the maximum packet size supported by the links between two endpoints without the fragmentation of an IP packet [36]. Packets with the "do not fragment" (DF) flag set are transmitted from source to destination. If a link along the path between the source and the destination does not support the packet size without fragmentation, the intermediate router returns an Internet Control Message Protocol (ICMP) [37] packet of type "destination unreachable". On receipt of the ICMP message, progressively smaller packets are sent until no ICMP message returns. The size of the final packet is chosen as the MTU. Path MTU discovery allows TCP to maximize the ratio of data to overhead bytes. Hence, a larger MTU value enables TCP senders to reach maximum throughput more rapidly because the increase of *cwnd* is determined by the number of received ACKs.

2.3.5 Performance enhancing proxies (PEPs)

PEPs [31] are employed to improve degraded TCP performance caused by link characteristics. PEPs are not intended for general use because they have an undesirable property of violating the TCP end-to-end semantics.

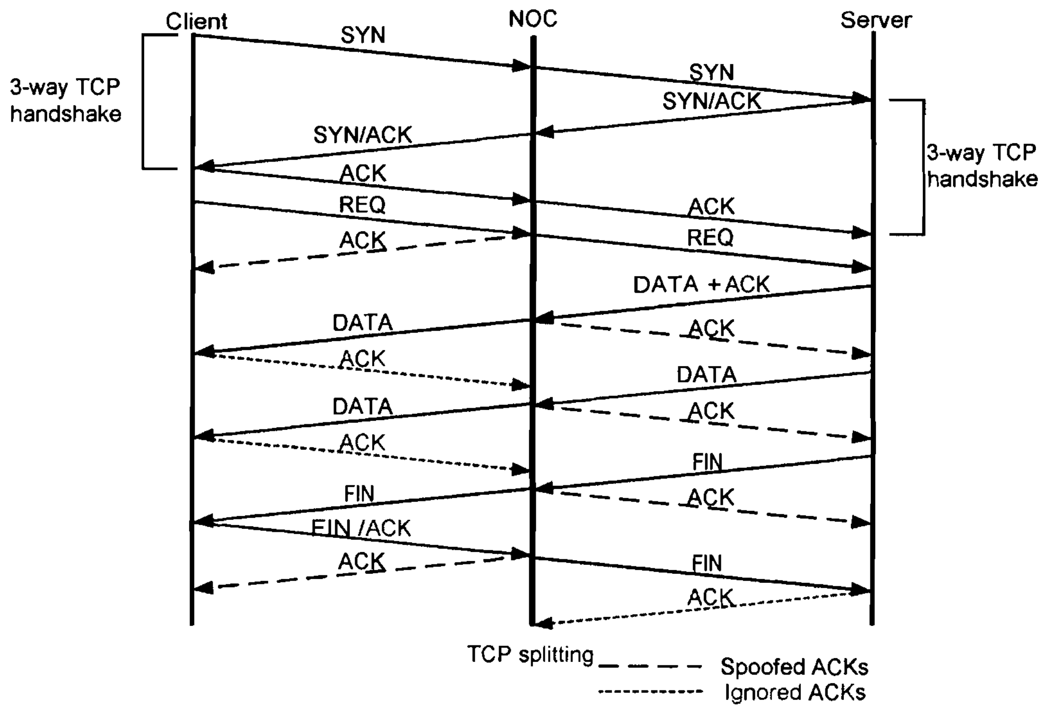

Figure 2.3. TCP splitting with spoofing.

The DirecPC system employs TCP splitting with spoofing in its PEPs, as illustrated in Figure 2.3 [38], [39]. A TCP connection is split at the NOC (centered vertical line), the intermediary between a satellite user (client) and a website (server). The three-way TCP handshake (SYN, SYN/ACK, and ACK) used in a split connection is identical to the handshake in the end-to-end TCP connections. However, subsequent TCP segments from both the client and the server (endpoints) are acknowledged by the NOC on behalf of the other endpoint using spoofed ACKs (dashed lines). ACKs transmitted by the two endpoints are ignored by the NOC (dotted lines). The spoofed ACKs from the NOC arrives at the two endpoints earlier compared to ACK arrivals in an end-to-end connection. TCP spoofing allows the *cwnd* to grow faster, resulting in improved performance.

TCP splitting with spoofing improves TCP performance in PEPs. However, it imposes considerable memory requirements at the NOC. All segments prematurely acknowledged by the NOC must be kept in local buffers until segments are acknowledged by the endpoints. As a consequence, the NOC is also responsible for retransmitting all lost segments.

2.4 Network anomalies

With the changing nature of network traffic, variety of network anomalies have emerged. Network anomalies refer to deviations from expected traffic patterns. Anomalies such as scans and worms are security threats. Others, such as denial of service (DoS) attacks and flash crowds, impact network performance. Hence, detection of network anomalies remains an important issue and is an open problem. The most common types of network anomalies are scans and worms, DoS, flash crowd, traffic shift, alpha traffic, and volume anomalies [15], [20], [40].

2.4.1 Scans and worms

There are two types of scans: network scans and port scans. Network scans are characterized by packets sent from a single host to probe network hosts at a single port. Port scans target multiple ports on host(s). In small networks, the two types of scans are difficult to distinguish. Unlike network scans, packets sent by worms originates from multiple hosts. In most instances, scans and worms are malicious in nature. They are used to discover and exploit network resources.

2.4.2 Denial of service

Denial of service (DoS) attacks and distributed denial of service (DDoS) attacks are characterized by a large number of packets from one or many hosts sent to a single destination. These packets may be TCP SYN packets (SYN flood), ICMP ping packets (ping flood), TCP RST packets (reflection attack), or other types of packets. The goal of DoS or DDoS is to render a host incapable of handling incoming connections or to exhaust the available network bandwidth along the network path(s) to the destination.

2.4.3 Flash crowd

Flash crowd refers to a high volume of traffic destined to a single IP. Flash crowds connections differ from DoS attacks because they are usually short-lived and only involve a single occurrence. Flash crowd events are often caused by the availability of certain information such as breaking news, product announcements, and new software releases.

2.4.4 Traffic shift

Traffic shifts occur when traffic is redirected from one set of paths to another. They are usually the consequence of route changes caused by link unavailability, network congestion, and routing table changes.

2.4.5 Alpha traffic

Alpha traffic refers to unusually high volume of traffic between a pair of endpoints resulting from bandwidth measurements or file transfers. Even though such traffic may not be malicious in nature, presence of alpha traffic increases the overall network delay and may cause network congestion.

2.4.6 Volume anomalies

Traffic volume anomalies include both outages and short-term increases in traffic demand. Outages are caused by the unavailability of network resources and can be caused by unavailable links, crashed servers, or routing problems. Short-term increases in traffic demand involve multiple sources or destinations and may be caused by short term demands such as activities during holidays. Both anomalies are characterized by a significant deviation of traffic volume from the usual daily or weekly patterns.

Chapter 3

Mathematical tools for statistical analysis

3.1 Cluster analysis background

Cluster analysis employs algorithms to group data objects into clusters based on their common characteristic(s) that may not be known. [41]–[43]. Similarity is the measure of the common characteristic(s), which is often defined as the distance between the objects. Objects are grouped into clusters by maximizing the intracluster similarity and minimizing the intercluster similarity. Hence, the goal of clustering is to group objects that have high similarity to each other within the same cluster (high intra-cluster similarity) while the objects in one cluster are dissimilar to objects in other clusters (low intercluster similarity). Cluster quality is the measure of the similarity within a cluster and dissimilarity between clusters. Two categories of clustering methods that we employed are partitioning methods and hierarchical methods.

3.1.1 Partition clustering

Partition clustering constructs k clusters (partitions) of the data from n objects where $k \leq n$. In partitioning methods, the objects are first divided among the k clusters. An iterative relocation technique is then used to improve the cluster quality. The partitioning algorithm has two constraints:

1. each cluster should contain at least one object and
2. each object should belong to exactly one cluster.

Exhaustive enumeration of all possible combinations is required to find the optimal clustering result. Hence, a heuristic method such the k-means algorithm is employed.

3.1.1.1 k-means clustering

The k-means algorithm [44] generates k clusters from n objects in d-dimensional space. They also require two inputs: k number of desired partitions and n objects. The goal of the clustering method is to produce clusters that have high intracluster similarity and low intercluster similarity. The cluster similarity in the k-means algorithm is measured by distance between the objects inside the cluster. The k-means algorithm is executed in the following steps:

1. k objects are selected randomly to be the centers of the k clusters.
2. Each remaining object is assigned to the most similar cluster. The similarity is measured by the distance between each object and the cluster mean.
3. The cluster mean is recalculated after all objects are (re)assigned.
4. All objects are re-evaluated and placed in the most similar cluster.
5. Steps 3 and 4 are repeated until no changes are made between iterations (full convergence) or until the maximum number of iterations is reached (partial

convergence).

The mean squared-error (MSE) is minimized upon convergence. The error function is computed as

$$E = \sum_{i=1}^{k} \sum_{p \in C_i} |p - m_i|^2, \tag{3.1}$$

where E is the sum of squared-error of all objects, p is the coordinate of an object and m_i is the mean of cluster C_i. There are several methods to compute the distance measure for the minimum MSE (MMSE) in multi-variable (multi-dimension) k-means clustering. The most common measure is the Euclidean distance, where the cluster centroid is calculated from the mean of all objects within a cluster in all dimensions. Another distance measure called *"cityblock"* is used for objects whose dimensions have no Euclidean distance relationships. This distance measure is the sum of absolute differences in each dimension. Hence, the centroid in the *"cityblock"* distance measure is the mean between the points for each dimension.

The k-means algorithm may only be applied to data sets that have a defined cluster mean. The computational complexity of the algorithm is $O(nkt)$, where n is the number of objects, k is the number of clusters, and t is the number of iterations. One of the challenges of the k-means algorithm is the need to define k *a priori*. However, the natural number of clusters is usually not known before the algorithm is applied. Furthermore, the k-means algorithm is sensitive to outliers. The choice of objects for the initial centers may also affect the clustering result because the k-means algorithm converges to the local minima. This can be mitigated by repeating the algorithm several times with different initial cluster centers and by selecting the best result through measuring cluster quality.

3.1.1.2 Measuring cluster quality and determining the number of clusters using silhouette coefficients

The k-means error function drops monotonically as the number of clusters k increases. Hence, cluster quality should be independent of k. Silhouette coefficients (SC) have been used to identify the quality of clustering results [45], [46]. The SC of an object i within a cluster A is calculated as follows:

1. Set a_i to be the average distance of object i to all other objects in cluster A.
2. For all clusters that is not A, calculate the average distance of object i to the objects in each of these clusters. Set b_i to be the minimum of these average distances and B to be the cluster with this minimum average distance to object i.
3. The silhouette coefficient s_i for object i is given by

$$s_i = \frac{b_i - a_i}{max(a_i, b_i)}. \tag{3.2}$$

The SC value lies between -1 and 1. An object i with a SC value of -1 is poorly clustered since object i is closer to objects in cluster B than it is to the objects in cluster A. On the contrary, SC value of 1 denotes a well-clustered object. An object with SC value of 0 can be grouped into either cluster A or B.

To evaluate cluster quality, the average SC value from all points in a cluster is calculated. The relationship between SC value and cluster quality is shown in Table 3.1. A cluster exhibits strong cluster quality if its average SC value is $0.7 <$ SC ≤ 1.0 [46]. Average SC value of $0.5 <$ SC ≤ 0.7 indicates medium cluster quality. Average SC $0.25 <$ SC ≤ 0.5 indicates low cluster quality. Average SC ≤ 0.25 indicates an absence of cluster structure. In k-means clustering, the natural number of clusters

Table 3.1. Cluster quality measured by average SC values [46].

Average SC value	Cluster quality
$0.7 < \text{SC} \leq 1.0$	Good
$0.5 < \text{SC} \leq 0.7$	Medium
$0.25 < \text{SC} \leq 0.5$	Low
$\text{SC} \leq 0.25$	Absence of cluster structure

can be found by plotting the average SC value versus k and by locating its local maximum.

3.1.2 Hierarchical clustering methods

Hierarchical clustering methods place objects into a hierarchical tree of clusters called a dendrogram. At the leaves of a dendrogram, each object is in its own cluster. At the top of a dendrogram, all objects belong to a single cluster. Hierarchical clustering can be divided into agglomerative (bottom-up) and divisive (top-down) approaches. In the agglomerative approach, each object begins in its own cluster. Successive steps merge into a cluster objects that are close to each other until all objects are merged into one cluster or the termination condition is reached. In contrast, the divisive approach starts with all objects belonging to a single cluster. In each successive iteration, clusters are divided into smaller clusters until each object belongs to its own cluster or the termination condition is reached. For both approaches, the most common termination condition is the number of desired clusters k. Another termination condition for the agglomerative (divisive) approach is to set the maximum (minimum) distance for merging (dividing) clusters.

Most hierarchical clustering methods employ the agglomerative approach. They differ in their definitions of intercluster similarity and optimizations employed to improve cluster quality. Four common intercluster distance measures are: minimum, maximum, mean, and average. For two clusters C_i and C_j, the distance between $p_i \in C_i$, $p_j \in C_j$ is $|p_i - p_j|$. The mean and the number of object for clusters C_i and C_j are m_i, m_j, n_i, and n_j, respectively. The four distance measures are defined as [41]:

- minimum distance (single linkage):

$$d_{min}(C_i, C_j) = \min_{p_i \in C_i, p_j \in C_j} |p_i - p_j|$$

- maximum distance (complete linkage):

$$d_{max}(C_i, C_j) = \max_{p_i \in C_i, p_j \in C_j} |p_i - p_j|$$

- mean distance (centroid linkage):

$$d_{mean}(C_i, C_j) = |m_i - m_j|$$

- average distance (average linkage):

$$d_{avg}(C_i, C_j) = \frac{1}{n_i n_j} \sum_{p_i \in C_i} \sum_{p_j \in C_j} |p_i - p_j|.$$

With the minimum distance measure, two clusters C_i and C_j are merged if the closest distance between p_i and p_j is the smallest. With the maximum distance measure, two clusters C_i and C_j are merged if the largest distance between p_i and p_j is the smallest. With the mean distance measure, the merge criterion is based on the smallest distance between the centroids of two clusters C_i and C_j. Finally, with the average distance measure, the merge criterion is based on the smallest of the average distance between all objects in the two clusters. A graphical illustration of the four distance measures is shown in Figure 3.1. The major challenge of hierarchical methods is that once a step (either merge or split) is performed, it cannot be reversed. An erroneous merge or split would result in a sub-optimal clustering result.

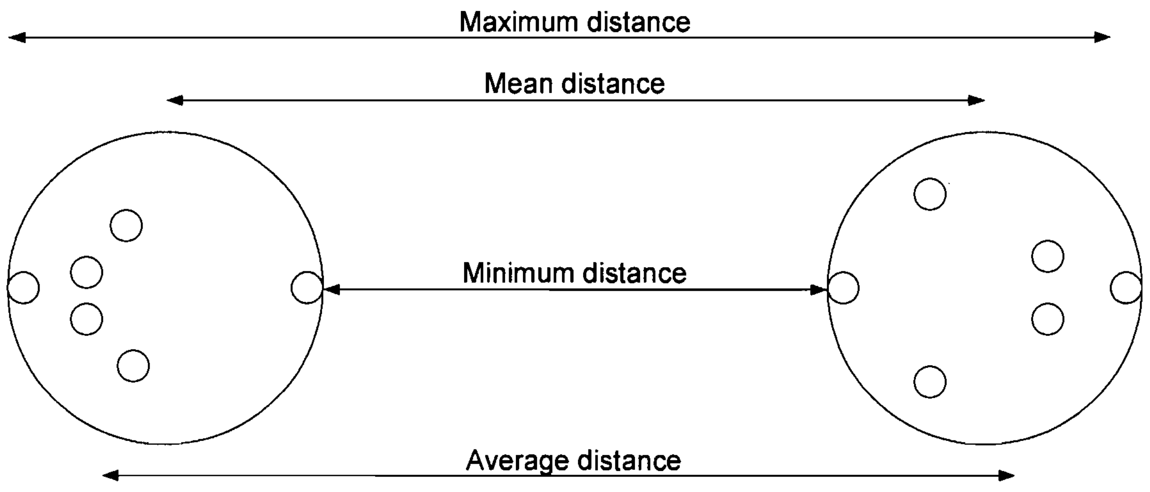

Figure 3.1. A graphical illustration of the four distance measures (minimum, maximum, mean, average) used in hierarchical clustering.

3.1.2.1 Implementation of agglomerative hierarchical clustering

Agglomerative hierarchical clustering is implemented in the following steps:

1. For n objects, a similarity matrix of size $n \times n$ is generated. The similarity matrix records the distance or, in the case of two number series, the number of identical points. The similarity matrix can be represented by a vector of size $\frac{n \times (n-1)}{2}$. For some implementations, a dissimilarity matrix is used. This matrix records the number of differences between two objects rather than the number of similarities.
2. Each of the n objects is assigned to clusters from 1 to n.

3. For each iteration, two objects most similar to each other (the largest similarity value or the smallest dissimilarity value) are merged into one cluster. A label is created for the new cluster that becomes the parent of the two child clusters. Hence, an object merged into a new cluster belongs to both the original cluster and the parent cluster.
4. The location of the centroid may change at the end of each iteration. Hence, if the mean distance measure is used, the similarity matrix is recomputed to reflect the creation of parent clusters.
5. Steps 3 and 4 are repeated until all objects are merged into a single cluster or the termination condition is reached.
6. Groups can be found by selecting either a desired number of clusters k or by selecting a maximum merge distance.

3.1.2.2 Visualization of hierarchical clustering

The results from hierarchical clustering can be visualized by plotting the dendrogram. An example of a dendrogram is shown in Figure 3.2. The merge distance between two objects is represented by the height of the link. Longer links indicate greater merge distances.

Groups can be determined using two methods. If the desired number of clusters k is chosen, the clusters can be determined by drawing a horizontal line such that the number of intersections between the line and the dendrogram is equal to k. Intersected links are removed. In Figure 3.2, a line is drawn for k=3. Each remaining binary tree is a cluster. A second method employs the inconsistency coefficient. This coefficient compares the height of a link in a cluster hierarchy with the average height of the links at the same level. Links connecting two distinct clusters have high inconsistency coefficient values whereas links connecting leaf clusters have values of zero. Links are removed if the inconsistency coefficient value is above a selected cutoff. Again, each remaining binary tree is labelled as a cluster. The inconsistency coefficient is defined as [47]:

$$\mathrm{IC} = \frac{Z_{ij} - \mu_{z\ considered}}{\sigma_{z\ considered}}, \tag{3.3}$$

where

- Z_{ij} = link distances between objects i and j in the hierarchical tree Z
- $\mu_{z\ considered}$ = mean of link distances considered in the calculation. Links considered are defined as links at the same level as Z_{ij} and links up to depth d below. The default value of d is 2.
- $\sigma_{z\ considered}$ = standard deviations of link distances considered in the calculation

3.1.2.3 Measuring cluster quality in hierarchical clustering methods

For hierarchical clustering, cluster quality can be visualized by plotting the similarity matrix. The matrix is first reorganized by ordering objects based on the cluster label at a chosen level. The matrix values are then normalized between 0 and 1, where a value of 1 indicates two objects as being identical. A value of 0 indicates two objects being entirely dissimilar. The normalized matrix value for an object i is calculated as

$$normalized\, matrix\, value_i = 1 - \frac{maximum\ similarity\, value - similarity\, value_i}{maximum\ similarity\, value}. \tag{3.4}$$

For well-clustered results, the plot of this matrix should be approximately block diagonal. Off-diagonal blocks indicate similarity between clusters.

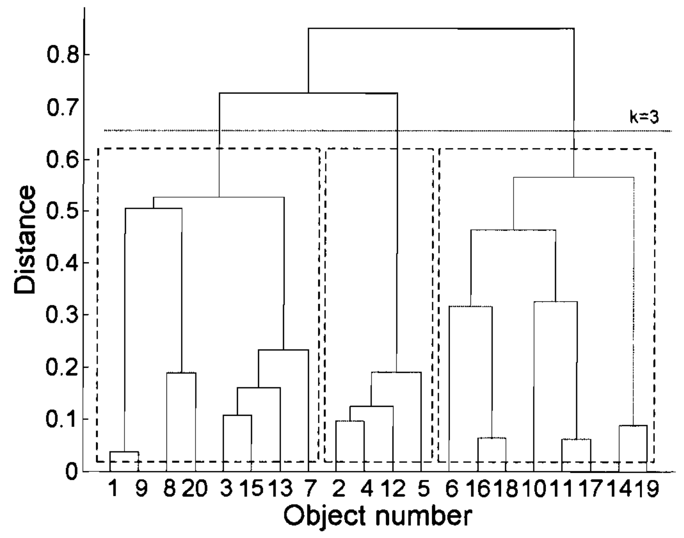

Figure 3.2. A dendrogram with 20 objects. The height of the links indicates the merge distances. The horizontal line intersects the dendrogram at 3 links, separating the dendrogram into 3 clusters, indicated by the 3 boxes.

The cophenetic correlation coefficient (CPCC) [41] is used to determine the best choice of distance measure for hierarchical clustering. CPCC is the correlation coefficient between the cophenetic distance matrix and the similarity matrix, where cophenetic distance is defined as the distance between two objects in the dendrogram to their common parent. CPCC is defined as [48]:

$$\mathrm{CPCC} = \frac{\sum_{i<j}(Y_{ij} - y)(Z_{ij} - z)}{\sqrt{\sum_{i<j}(Y_{ij} - y)^2 \sum_{i<j}(Z_{ij} - z)^2}}, \tag{3.5}$$

where

- Y = the actual distances between objects, Z is distances between objects in the hierarchical tree
- Y_{ij} = distances between objects i and j in Y
- Z_{ij} = distances between objects i and j in Z

- y = average distance of all of objects in Y and
- z = average distance of all objects in Z.

If the distance between two merged clusters is 0.1, the cophenetic distance between all points within one cluster to the points in the second cluster is also 0.1. A higher correlation coefficient indicates better clustering results.

3.2 Wavelet analysis

Wavelet transforms have been used to evaluate non-stationary signals [49]–[51]. They are employed to decompose a signal into different time scales, enabling analysis in both time and frequency domains. In comparison, the common Fourier transform discards the locality information from the time domain and cannot accurately reconstruct non-stationary signals.

The discrete wavelet transform (DWT) is used to analyze discrete signals such as data traffic. The most common type of DWT is the dyadic, where signals are sampled in powers of two. The dyadic DWT is defined as

$$\begin{aligned} d_{j,k} &= \int_{\infty}^{\infty} X(t) 2^{-\frac{j}{2}} \psi\left(2^{-j}t - k\right) dt && (3.6) \\ &= \int_{\infty}^{\infty} X(t) \psi_{j,k}(t) dt, && (3.7) \end{aligned}$$

where $d_{j,k}$ is the wavelet coefficient at scale level j and translation k, $X(t)$ is the original signal, and $\psi_{j,k}(t)$ is the basis function of the transform. The series of wavelet coefficients at scale level j is referred to as the detail coefficients d_j. The basis function $\psi_{j,k}(t)$ is obtained by dilating the mother wavelet $\psi(t)$ by a factor of j and translating (time shifting) by k time units. The relationship between the mother wavelet $\psi(t)$

and the basis function $\psi_{j,k}(t)$ is

$$\psi_{j,k}(t) = \frac{1}{\sqrt{j}}\psi\left(\frac{t-k}{j}\right), j \in \mathbb{R}^+, k \in \mathbb{R}. \tag{3.8}$$

The inverse DWT is defined as

$$X(t) = \sum_{j=0}^{\infty}\sum_{k=-\infty}^{\infty} d_{j,k}\psi_{j,k}(t). \tag{3.9}$$

Equation (3.9) can be rewritten as

$$\begin{aligned} X(t) &= \sum_{j=l+1}^{\infty}\sum_{k=-\infty}^{\infty} d_{j,k}\psi_{j,k}(t) + \sum_{j=0}^{l}\sum_{k=-\infty}^{\infty} d_{j,k}\psi_{j,k}(t) && (3.10)\\ &= \sum_{k=-\infty}^{\infty} a_{l,k}\phi_{l,k}(t) + \sum_{j=0}^{l}\sum_{k=-\infty}^{\infty} d_{j,k}\psi_{j,k}(t). && (3.11)\end{aligned}$$

In (3.10) and (3.11), the sum over j is divided into two regions. The first summation is an approximation of the original signal $X(t)$ at scale level l and refers to the approximation coefficients a_l. The number of coefficients in the approximation is $\frac{\text{length of } X(t)}{2^l}$. Higher values of l indicate coarser approximations. The function $\phi_{l,k}(t)$ is the scaling function at scale level l. The second summation is the sum of the details at the scale level l.

Chapter 4

Analysis of billing records

We analyze two months of billing records collected from the DirecPC system. The records contain a collection of hourly-generated files with information about traffic volume in terms of packets and bytes. These billing records capture the hourly network dynamics.

4.1 Data format

DirecPC billing records contain hourly summary of satellite user activities. We use 1,688 and 1,704 files of billing record collected from two hosts Turbo1 and Turbo2, respectively. The hosts were located at the NOC. Each file records the activity of DirecPC users during a particular hour and has the file extension **.bil.* The records are collected continuously every hour from 23:00 on Oct. 31, 2002 to 11:00 on Jan. 10, 2003. In total, 1,691 hours of billing records were collected. The hourly activities for a satellite user are usually recorded in one of the two hosts. However, if a satellite user disconnects from the network and reconnects during an hour, a non-overlapping record for the user may also be present on the second host. The files contain information about satellite user activities using the eleven fields shown in Table 4.1 [52]. The field names for each entry are: *RecLen*, *RecTyp*, *SiteID*, *Start*, *Stop*, *Cmin*, *Bill*, *CTxByt*, *CRxByt*, *CTxPkt*, and *CRxPkt*. We are interested in *SiteID*, *Start*, *CTxByt*, *CRxByt*, *CTxPkt*, and *CRxPkt*. *SiteID* is a unique hexadecimal ID for each user, *Start* is the hourly timestamp while *CTxByt*, *CRxByt*, *CTxPkt*, and *CRxPkt* summarize the number of bytes and packets for each direction of data transfer. In the billing records, *Tx* refers to the traffic sent by the NOC to a user through the satellite link while *Rx* refers to the traffic sent by the user to the NOC through terrestrial dial-up modem. We refer to the *Tx* direction as download and the *Rx* direction as upload with respect to a user.

4.2 Data pre-processing

We use MATLAB [53] to analyze the billing records. Prior to the analysis, we combined the files, remove the invalid entries, and then merge the remaining entries. The first step is to combine all entries from the **.bil* files into a single file using a Linux *Bash* script file *combine.sh*. The script concatenates all **.bil* files collected from Turbo1 and Turbo2 into a single file. We then examine the file manually to remove invalid entries. Invalid entries occur when a billing file is empty and the text "[MMDDhhmm]: writing billing records" appear instead of an entry (MMDDhhmm refers to the month, date, hour, and minutes). The lines containing these texts are manually deleted before the next processing stage. The fields in the file are tab-delimited. Hence, the script file *delimit.sh* is used to change the tab-delimited file to

Table 4.1. DirecPC billing record format.

Field name	Field Length (without delimiter)	Description
RecLen	5	Length of record including new line character (value is 00100)
RecTyp	3	DirecPC record type (value is 001)
SiteID	10	Identifies a subscriber by an unique alphanumeric string
Start	14	Start time of the call record with format YYYYMMDDhhmmss (20021031230007)
Stop	14	Stop time of the call record with format YYYYMMDDhhmmss (20021101000007)
Cmin	3	Number of active minutes (minutes in the recorded period during which the subscriber transmitted or received packets)
Bill	1	Identifies a subscriber's dial-up method
CTxByt	10	Number of bytes the NOC has transmitted to the subscriber through the satellite link (number of downloaded bytes the subscriber has received using the satellite link)
CRxByt	10	Number of bytes the NOC has received from the subscriber (number of uploaded bytes the subscriber has transmitted to the NOC using dial-up)
CTxPkt	10	Number of packets the NOC has transmitted to the subscriber through the satellite link (number of downloaded packets the subscriber has received using the satellite link)
CRxPkt	10	Number of packets the NOC has received from the subscriber (number of uploaded packets the subscriber has transmitted to the NOC using dial-up)

the comma-delimited format and convert the hexadecimal siteID to base 10. These changes are required to facilitate data import into MATLAB.

The *normalize.m* function that we developed is used to convert the hourly records from the YYYYMMDDhhmmss format to hours numbered from 0 to 1691 (YYYYMMDDhhmmss refers to year, month, hour, minute, and second). The last pre-processing step is performed through the *mergebilling.m* function. This function sorts the entries according to the hour and the SiteID and combines two entries if their hour and *SiteID* are identical.

4.3 Analysis of the aggregated traffic

After pre-processing, we aggregate the billing records by hour and by day. For each hour and day, four values are recorded: downloaded bytes, uploaded bytes, downloaded packets, and uploaded packets.

4.3.1 Hourly and daily traffic volume

The aggregated downloaded and uploaded hourly and daily traffic data in terms of packet and bytes are shown in Figures 4.1 – 4.8. The downloaded traffic (bytes) is higher than the uploaded traffic (bytes) by an order of magnitude. Uploaded number of packets is only slightly larger compared to downloaded number of packets because sent requests are usually followed by a received response. The difference may be attributed to the presence of the User Datagram Protocol (UDP) packets because UDP does not require acknowledgement for packets sent, unlike TCP. A regular pattern that repeats every 24 hours is observed in Figures 4.1 – 4.4. An exception occurs on Dec. 24, 2002, when the daily minimum traffic volume is much higher compared to other daily minima. On Jan. 3, 2003, the traffic volume decreased to almost zero followed by the highest recorded traffic volume, as shown in Figures 4.1 and 4.2. This change in the traffic pattern was caused by a network outage followed by the transmission of queued data after recovery. The maximum number of downloaded packets is recorded on Dec. 24, 2002, as shown in Figures 4.5 and 4.6, indicating the change in traffic dynamics during holidays. We also observed a drastic reduction in traffic volume during the extended holiday season between Jan. 1, 2003 and Jan. 10, 2003, as shown in Figures 4.5 – 4.8.

4.3.2 Daily (diurnal) and weekly cycles

Daily and weekly cycles are observed by averaging the data traffic for the same hour over all days or over the same day of a week. The daily cycles for packets and bytes are shown in Figures 4.9 and 4.10, respectively. The weekly traffic averages for packets and bytes are shown in Figures 4.11 and 4.12, respectively.

A daily minimum appears at 7 AM. The data traffic volume then rises rapidly until it reaches daily maxima at 11 AM, 3 PM, and 7 PM. The traffic volume decreases monotonically from 7 PM until 7 AM. Similar traffic patterns have been reported [54], with the third daily maximum occurring later in the evening (between 9 PM and 10 PM) rather than at 7 PM. As expected, the traffic volume on weekends is lower than on working weekdays. The three daily maxima for Wednesdays are not as visible as for other days, as shown in Figure 4.12, because both Dec. 24 and Dec. 31, 2002 occur on a Wednesday. This suggests that traffic volume may have different patterns on days immediately prior to holidays.

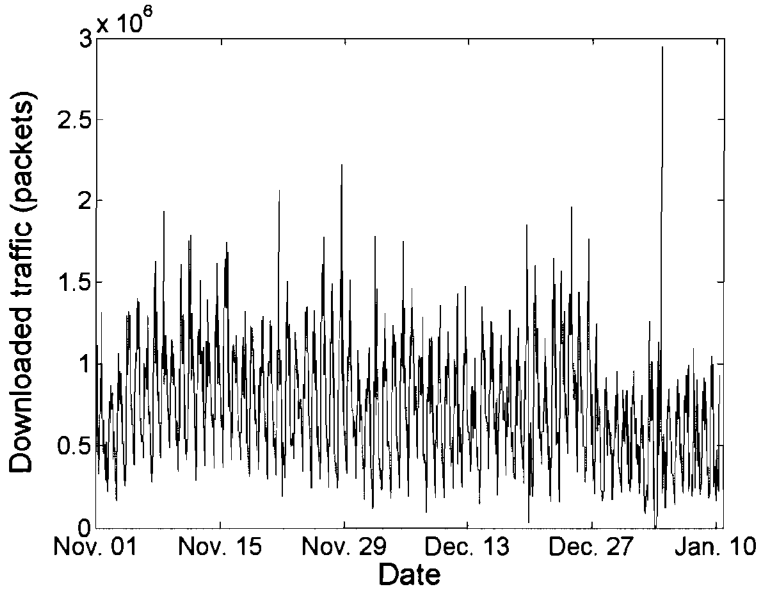

Figure 4.1. Aggregated traffic: downloaded packets (hourly). The data was recorded from 23:00 on Oct. 31, 2002 to 10:00 on Jan. 10, 2003. The highest packet volume was recorded on Jan. 3, 2003.

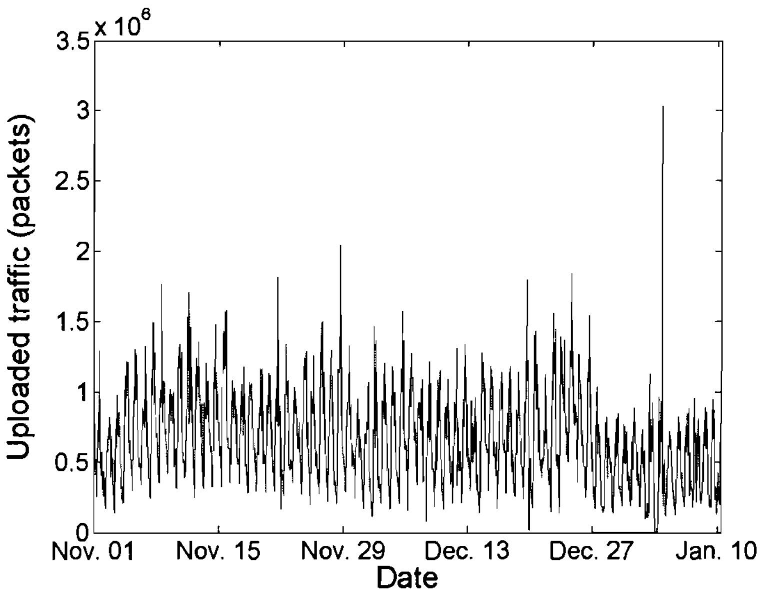

Figure 4.2. Aggregated traffic: uploaded packets (hourly). The data was recorded from 23:00 on Oct. 31, 2002 to 10:00 on Jan. 10, 2003. Highest packet volume was recorded on Jan. 3, 2003.

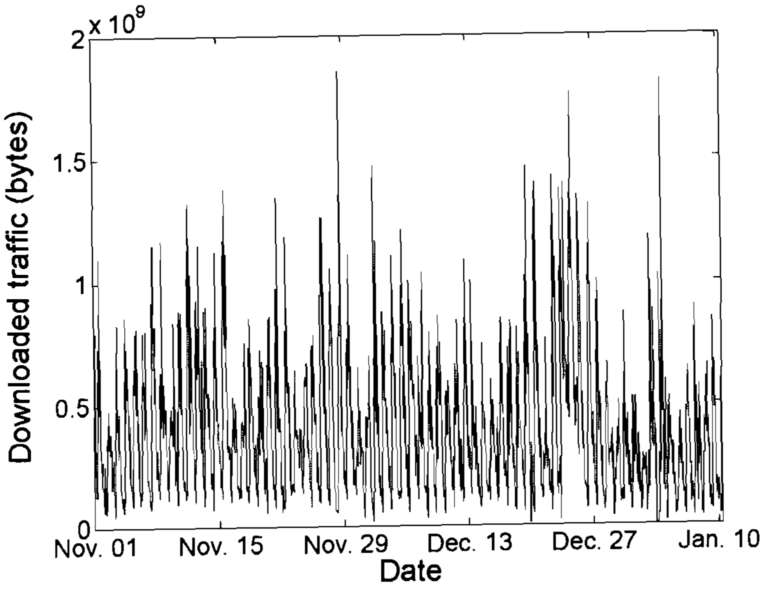

Figure 4.3. Aggregated traffic: downloaded bytes (hourly). Highest daily minimum downloaded traffic volume was recorded on Dec. 24, 2002.

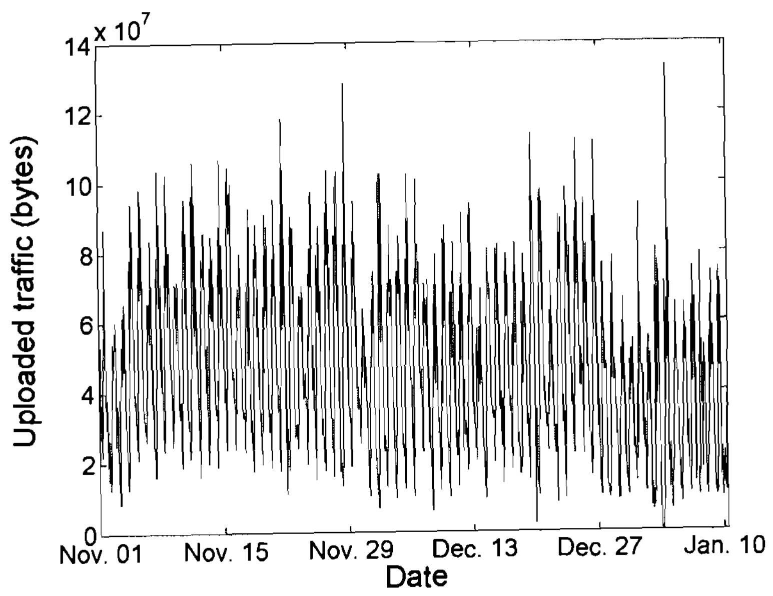

Figure 4.4. Aggregated traffic: uploaded bytes (hourly). The uploaded bytes is one order of magnitude smaller than downloaded bytes.

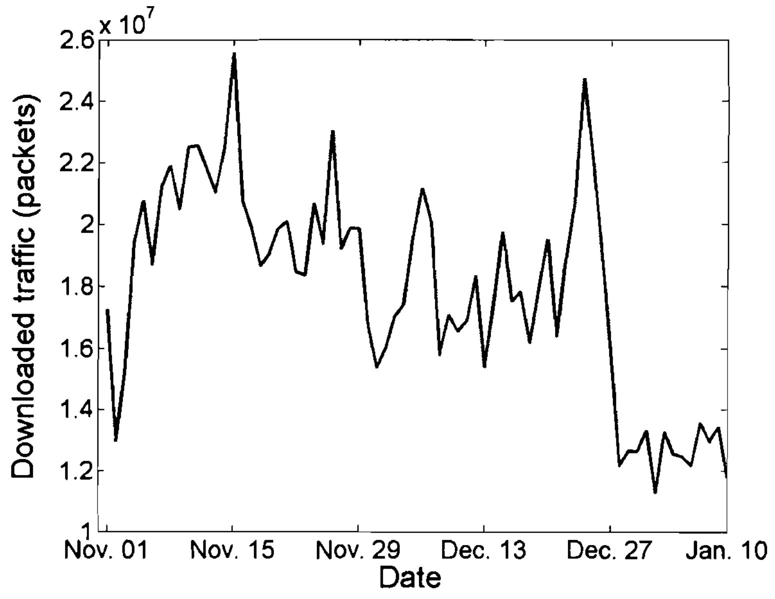

Figure 4.5. Aggregated traffic: downloaded packets (daily). Downloaded packet volume is lower between Jan. 1 and Jan. 10 compared to the rest of the recorded period.

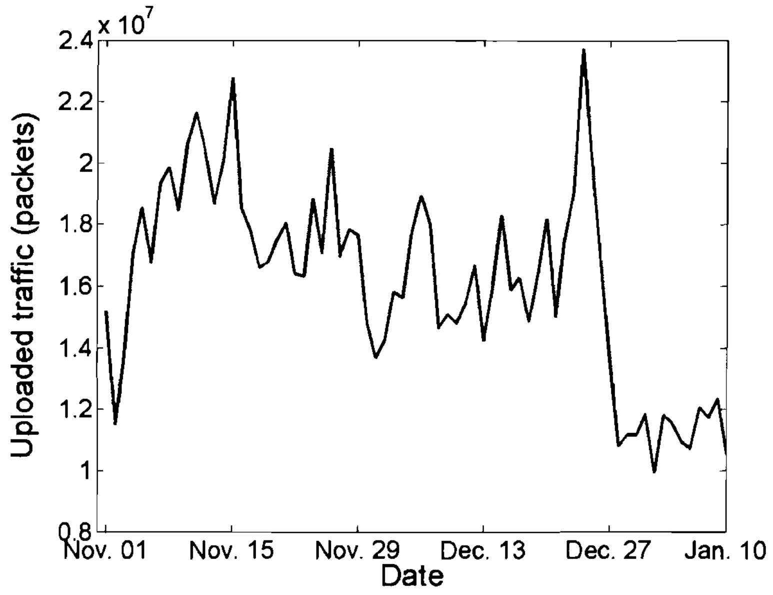

Figure 4.6. Aggregated traffic: uploaded packets (daily). Uploaded packet volume is lower between Jan. 1 and Jan. 10 compared to the rest of the recorded period.

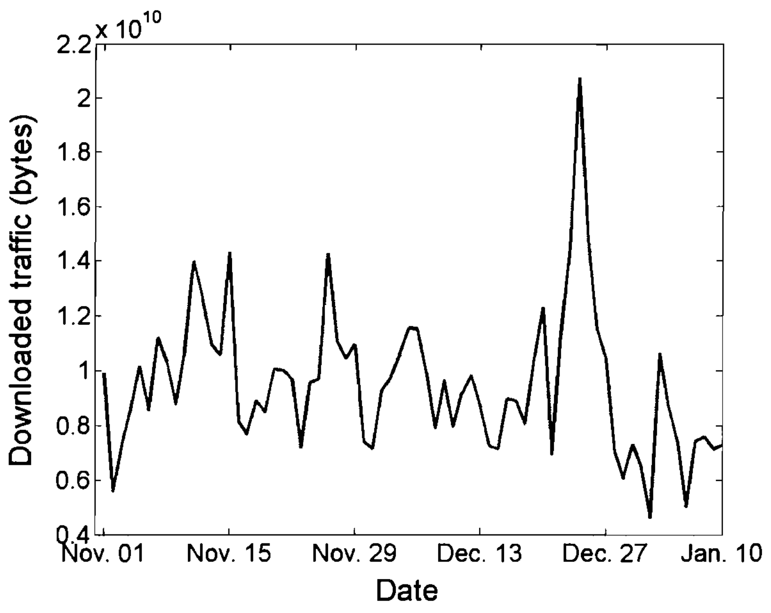

Figure 4.7. Aggregated traffic: downloaded bytes (daily). The highest number of daily downloaded bytes was recorded on Dec. 24, 2002.

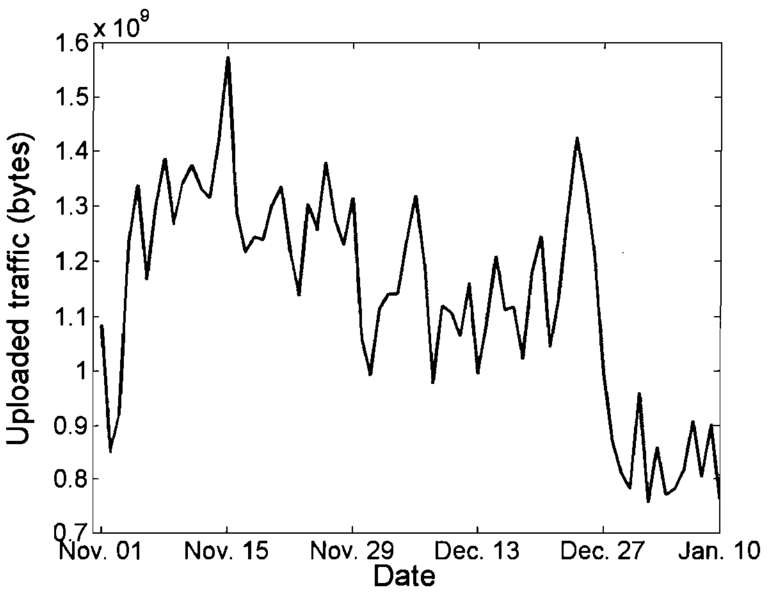

Figure 4.8. Aggregated traffic: uploaded bytes (daily). The highest number of daily uploaded bytes was recorded on Nov. 13, 2002.

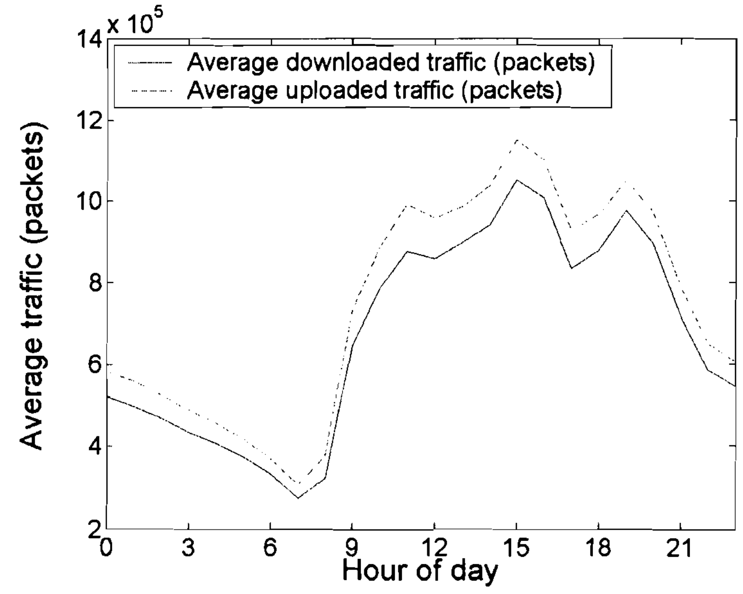

Figure 4.9. Average daily downloaded and uploaded traffic volume in packets obtained by averaging all recorded values for the same hour. The data was recorded from 23:00 on Oct. 31, 2002 to 10:00 on Jan. 10, 2003.

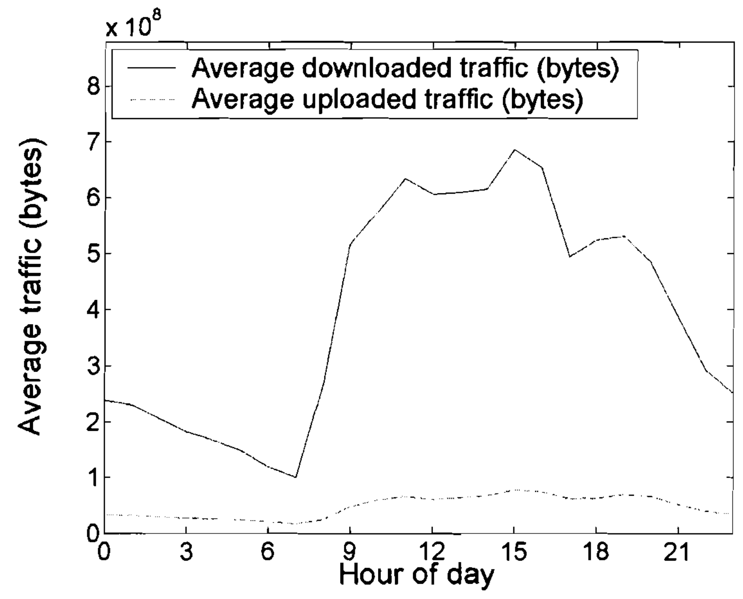

Figure 4.10. Average daily downloaded and uploaded traffic volume in bytes obtained by averaging all recorded values for the same hour. Average number of uploaded bytes is one order of magnitude smaller than average downloaded bytes.

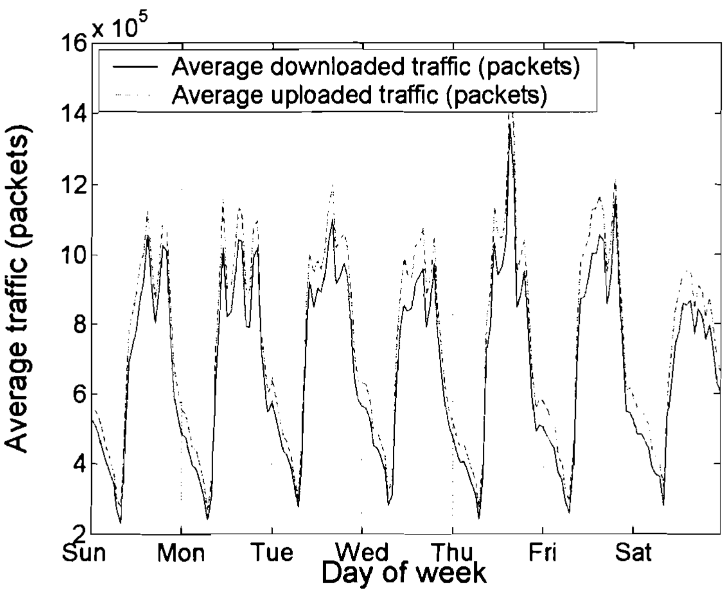

Figure 4.11. Average weekly downloaded traffic volume in packets obtained by averaging all recorded values for the same hour on the same weekday.

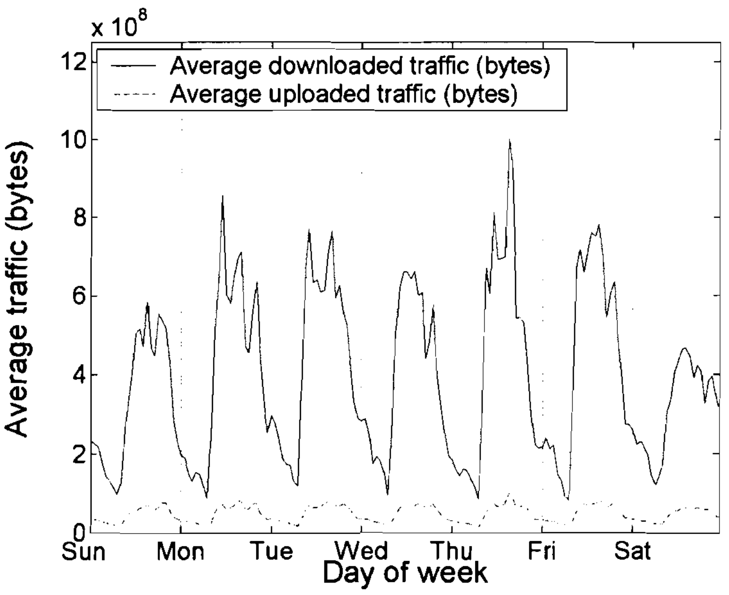

Figure 4.12. Average weekly downloaded traffic volume in bytes obtained by averaging all recorded values for the same hour on the same weekday.

4.4 Analysis of user behavior

In Section 4.3.2, we examined the aggregated records. While aggregated records enable us to observe the general characteristics of the traffic, the aggregation process discards information about individual users. In this Section, we investigate the traffic patterns of the 186 users identified in the ChinaSat billing records. We examine the traffic volume contributed by each user through ranking and we construct cumulative distribution functions (CDFs). We then classify the satellite users into distinct groups using cluster analysis. k-means cluster analysis is employed to classify users according to their average traffic per hour. Hierarchical clustering is employed to classify users based on their traffic patterns. We then refine the hierarchical clustering results by classifying users into the three most common traffic patterns.

4.4.1 Ranking of user traffic

We sum the total number of packets and bytes of each ChinaSat user. From the billing records, the user with the most traffic downloaded received 78.8 GB and uploaded 11.9 GB during the recorded period. The same user also downloaded/uploaded the most number of packets ($\sim$205 million packets).

We rank the users according to their total traffic in descending order in terms of downloaded and uploaded packets and bytes. The user who contributed the most traffic is placed at rank 1. The user ranks according to downloaded bytes are shown in Figure 4.13. Uploaded bytes and downloaded and uploaded packets have ranks that exhibit similar patterns. A user may not have the same rank across the four traffic statistics. Table 4.2 lists the ranks of the first 20 users ordered by decreasing number

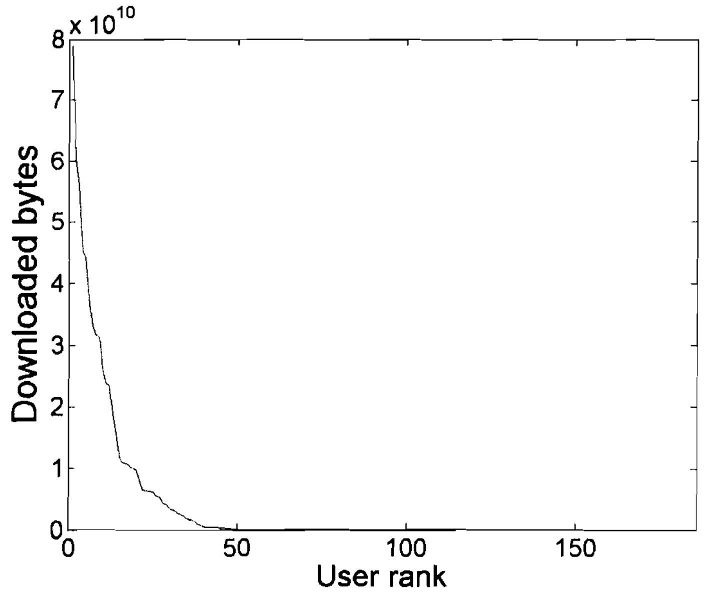

Figure 4.13. Users ranked by downloaded bytes. The user with the most downloaded bytes has rank equal to 1.

of downloaded bytes. Even though the user with the heaviest traffic is ranked 1 in all four rankings, the user with the second most downloaded bytes ranked only eighth in terms of the number of downloaded/uploaded packets.

From the ranks, we construct CDFs according to the traffic volume of each user. The results for downloaded traffic is shown in Figure 4.14. The top user accounts for 11% and the top 25 users account for 93.3% of the total downloaded bytes. The top 37 users contributed 99% of the total traffic. The four CDFs are shown in Figure 4.15. Although the CDF curves differ slightly, the distributions of downloaded and uploaded packets and bytes are very similar.

We use histograms to examine the relationship between traffic volume and number of users. The histogram for downloaded bytes is shown in Figure 4.16. More than 150 users downloaded less than 0.5×10^{10} bytes. The remaining 36 users downloaded

Table 4.2. Ranking of the 20 SiteIDs with the most downloaded bytes. The users are listed by the order of downloaded bytes. Rank 1 refers to the user with the highest number of downloaded bytes.

SiteID	Downloaded packets	Uploaded packets	Downloaded bytes	Uploaded bytes
71910659	1	1	1	1
128651268	8	8	2	13
73249794	7	11	3	8
72721924	4	4	4	4
72783107	11	9	5	11
71908356	2	2	6	3
73252098	3	3	7	2
72928519	5	5	8	6
73245697	10	10	9	10
72805121	9	7	10	5
72754948	6	6	11	7
73257474	13	13	12	9
72691714	17	17	13	17
72718850	12	12	14	12
72535041	18	19	15	16
71749895	21	21	16	18
72721923	16	15	17	15
73252102	14	14	18	14
72177156	19	18	19	24
73132546	22	23	20	20

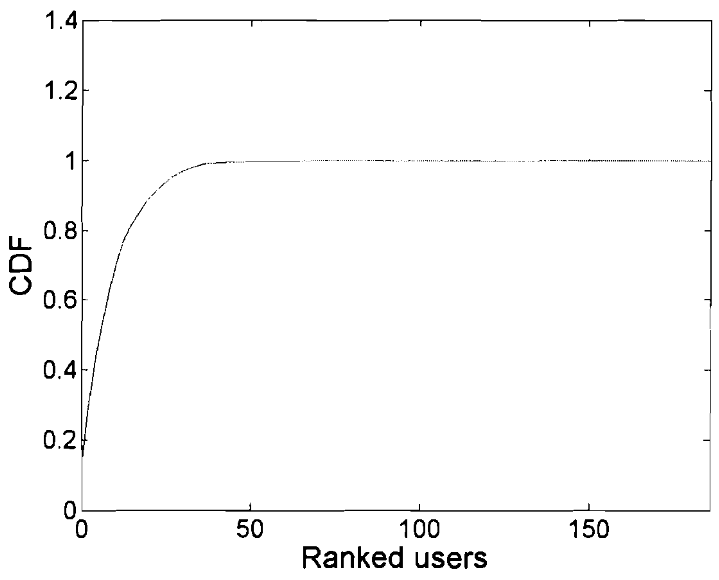

Figure 4.14. CDF of downloaded bytes. 99% of total traffic is contributed by the top 37 users.

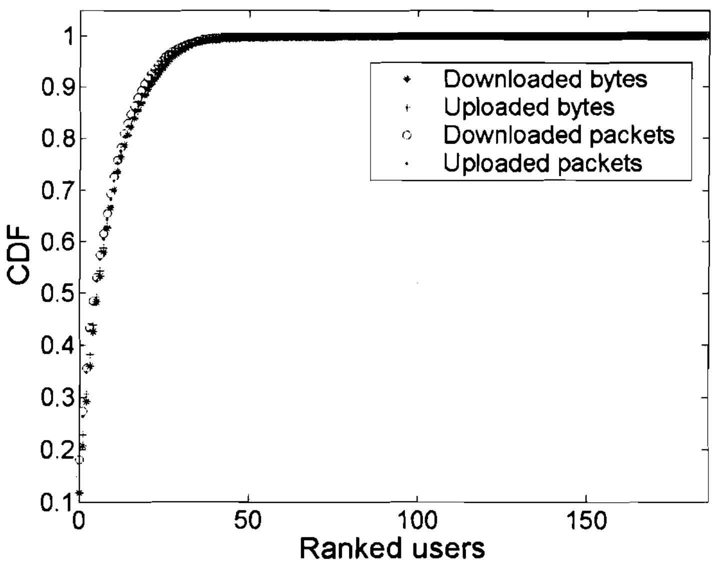

Figure 4.15. CDFs of downloaded and uploaded packets and bytes. The CDFs are very similar.

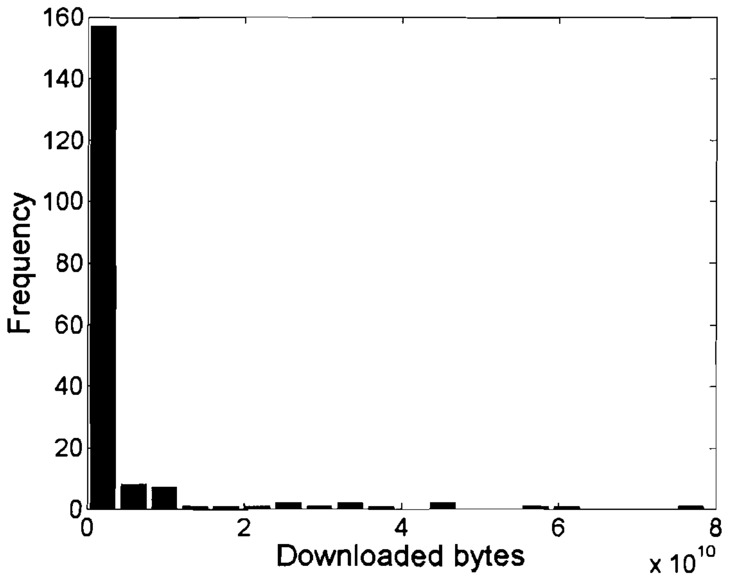

Figure 4.16. Histogram of the downloaded traffic. Majority of users (150) downloaded less than 0.5×10^{10} bytes.

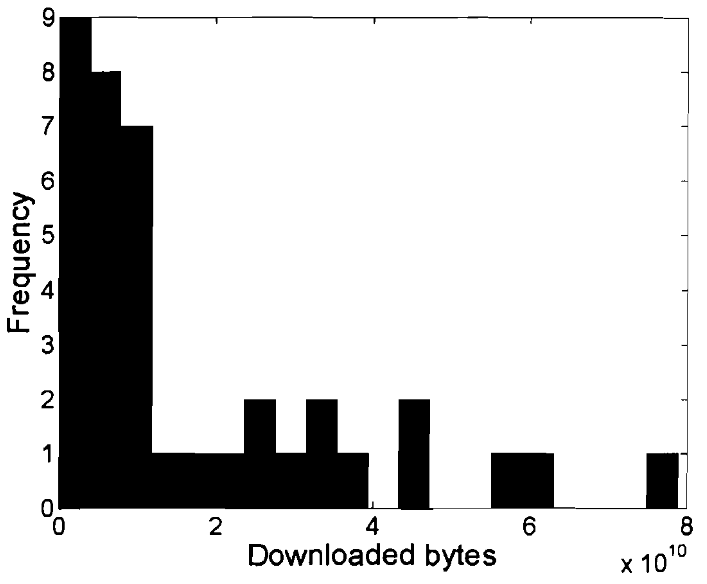

Figure 4.17. Histogram of the downloaded traffic for the 36 users who downloaded $> 0.5 \times 10^{10}$ bytes. With the exception of the first two bins, most bins have only two or fewer users.

between 0.5×10^{10} and 8×10^{10} bytes. The histogram for the 36 users with the most number of downloaded bytes is shown in Figure 4.17. The bars with higher number of users correspond to smaller number of downloaded bytes, as shown in Figure 4.17. However, the number of data points is insufficient to model the total user downloaded bytes using statistical distributions.

4.4.2 Classification of users with *k*-means clustering

We employ the k-means clustering method described in Section 3.1.1.1 to classify the ChinaSat users. We choose the MATLAB k-means implementation for our analysis. For the single-variable k-means clustering, we group the users according to the average traffic they contributed. For the multi-variable clustering, we combine the results from the single-variable clustering.

4.4.2.1 single-variable *k*-means clustering

Prior to employing the k-means algorithm, we first find the average packets and bytes downloaded/uploaded by each user. We choose to use average traffic per hour instead of total traffic because not all users are active through the entire period when the billing records were captured. For example, the user with *SiteID* 72721924 was only active between Nov. 23, 2002 and Jan. 10, 2003, as shown in Figure 4.18. This user contributed the fourth most downloaded and uploaded packets and bytes, as recorded in Table 4.2. Thus, if we use total traffic as the metric, a heavy traffic user who was active for only part of the recorded period may be misclassified as a medium traffic user. Hence, the average traffic would better serve our goal of classifying users.

We do not know *a priori* the natural number of groups that classifies the collected

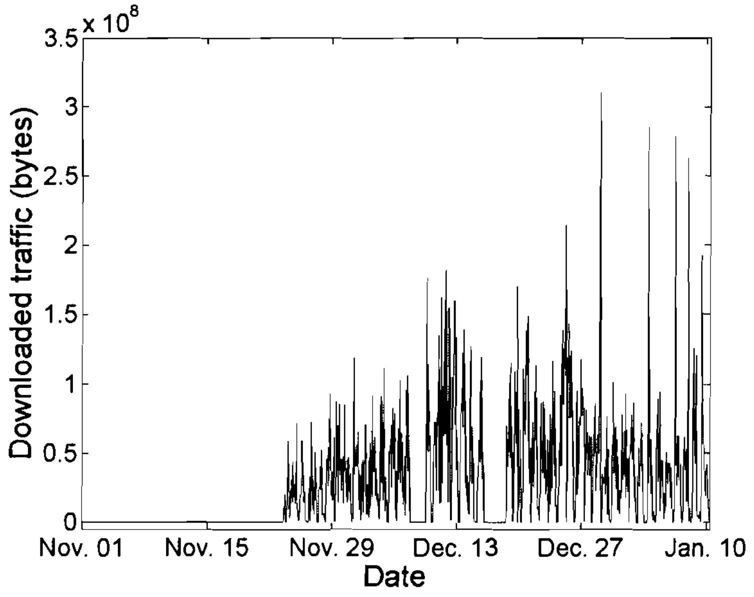

Figure 4.18. Downloaded bytes of user with *SiteID* 72721924. The vertical axis is offset to show the absence of traffic between June 30 and Nov. 23, 2002. Even though this user was one of the top traffic contributors, the user was only active from Nov. 23, 2002 to Jan. 10, 2003.

data. Hence, we tested k from 2 to 10. To mitigate the problem of empty clusters, MATLAB is configured to create a new cluster consisting of the one object furthest from its centroid. Furthermore, to avoid converging to a local minimum, we repeat the algorithm 15 times for each value of k using different sets of initial objects. We set the number of iterations to be 50,000 to ensure full convergence. Finally, we use silhouette coefficients to quantify the cluster quality.

All runs of the k-means clustering algorithm were completed within 50 iterations and within 2 minutes to full convergence. The average SCs from the cluster analysis of downloaded and uploaded packets are shown in Tables 4.3 and 4.4 and Figures 4.19 and 4.20, respectively. The natural number of clusters for both downloaded and uploaded packets is 2.

Table 4.3. Average SC for k-means clustering of downloaded packets.

k	Average silhouette coefficient (downloaded packets)
2	0.92729
3	0.92504
4	0.87268
5	0.8736
6	0.89665
7	0.89697
8	0.8926
9	0.87825
10	0.81644

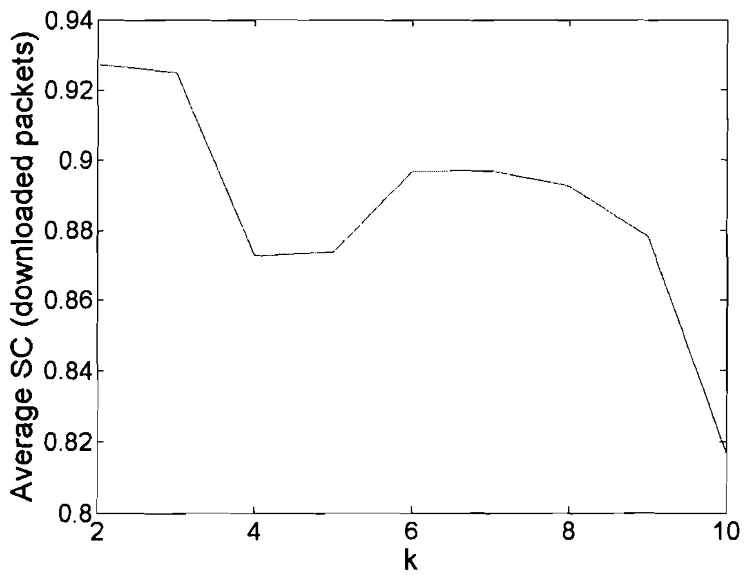

Figure 4.19. Plot of the average SC and k for downloaded packets. The natural number of clusters for a set of objects correspond to the local maxima of the average SC.

Table 4.4. Average SC for k-means clustering of uploaded packets.

k	Average silhouette coefficient (uploaded packets)
2	0.92534
3	0.91675
4	0.91256
5	0.8754
6	0.90034
7	0.90433
8	0.81919
9	0.90045
10	0.77038

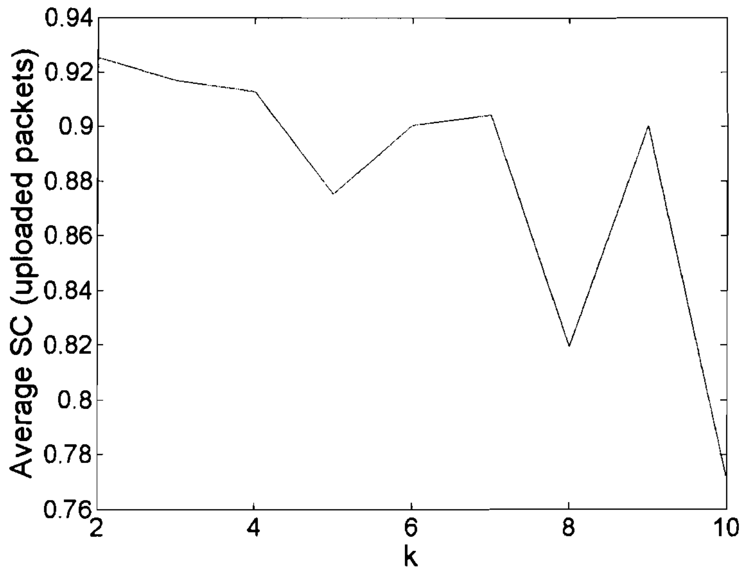

Figure 4.20. Plot of the average SC and k for uploaded packets. The natural number of clusters for a set of objects correspond to the local maxima of the average SC.

The k-means clustering results for downloaded and uploaded bytes differ from downloaded and uploaded packets, as shown in Tables 4.5 and 4.6 and Figures 4.21 and 4.22, respectively. The natural number of clusters for downloaded and uploaded bytes is 3. By examining the cluster boundaries, we refer to the three clusters as heavy, medium, and light traffic users.

Table 4.5. Average SC for k-means clustering of downloaded bytes.

k	Average silhouette coefficient (downloaded bytes)
2	0.89442
3	0.92996
4	0.89852
5	0.91555
6	0.89345
7	0.89431
8	0.8939
9	0.83582
10	0.8541

Table 4.6. Average SC for k-means clustering of uploaded bytes.

k	Average silhouette coefficient (uploaded bytes)
2	0.92134
3	0.92805
4	0.89049
5	0.89382
6	0.90428
7	0.90256
8	0.75551
9	0.80786
10	0.76943

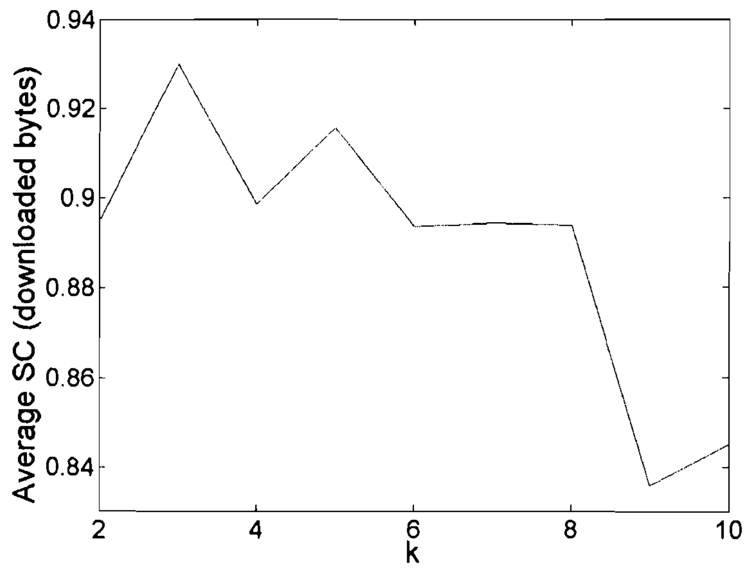

Figure 4.21. Plot of the average SC and k for downloaded bytes. The natural number of clusters for a set of objects correspond to the local maxima of the average SC.

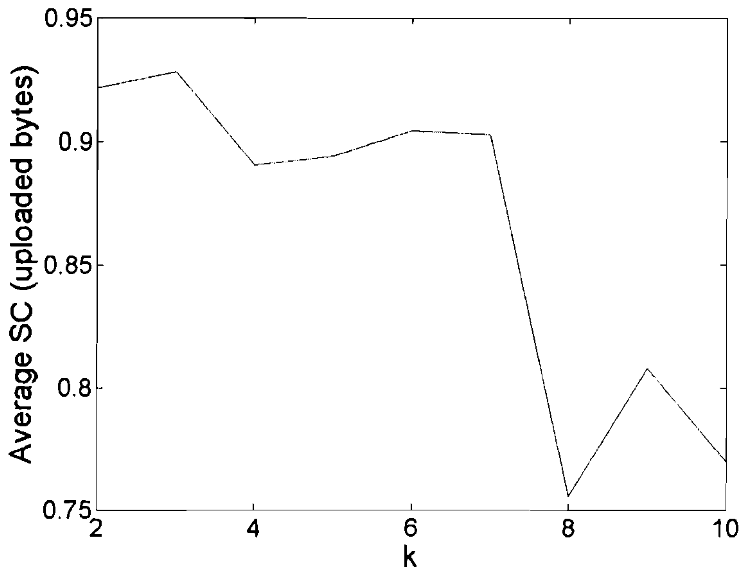

Figure 4.22. Plot of the average SC and k for uploaded bytes. The natural number of clusters for a set of objects correspond to the local maxima of the average SC.

We also examine the results for downloaded bytes. We list the cluster size, object boundaries, and average SC for each of the k clusters for downloaded bytes in Tables 4.7 and 4.8. Cluster 1 contains users who contributed the least amount of traffic while the cluster with the largest cluster number contains users who contributed the most volume of traffic.

The SC plot for each value of k is shown in Figures 4.23 - 4.31. Note that the SC plots for k=6 and 7 have no negative SC values. The lack of negative SC values suggests that k=6 and 7 may also be natural number of clusters. Nevertheless, the lower average SC values indicate that objects clustered using k=6 and k=7 are clustered worse compared to k=3 even though all objects have positive SC values.

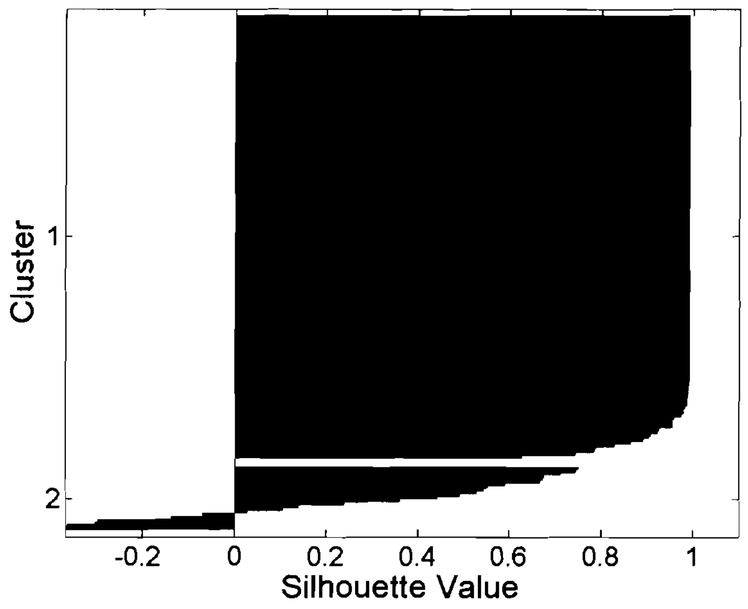

Figure 4.23. Silhouette plot: average downloaded bytes for k=2.

Table 4.7. Downloaded bytes: k-means clustering for k=2–7.

k	Cluster number	Cluster size	Lower cluster boundary (MB)	Upper cluster boundary (MB)	Cluster silhouette coefficient
2	1	163	0.0	23.3	0.97948
	2	23	23,3	110.7	0.29158
3	1	159	0.0	16.8	0.9703
	2	24	16.8	70.7	0.6706
	3	3	70.7	110.7	0.8668
4	1	147	0.0	7.2	0.96424
	2	16	7.2	21.1	0.73935
	3	20	25.4	60.5	0.5499
	4	3	80.8	110.7	0.85138
5	1	147	0.0	6.8	0.96424
	2	16	7.5	23.3	0.69893
	3	15	23.3	40.3	0.8044
	4	5	40.3	70.7	0.62392
	5	3	70.7	110.7	0.72696
6	1	135	0.0	3.5	0.97148
	2	17	3.5	10.7	0.6062
	3	11	10.7	23.3	0.72579
	4	15	23.3	40.3	0.76279
	5	5	40.3	70.7	0.62392
	6	3	70.7	110.7	0.72696
7	1	118	0.0	1.4	0.96437
	2	24	1.4	5.1	0.22518
	3	10	5.1	10.8	0.60267
	4	11	10.8	23.3	0.59385
	5	15	23.3	40.3	0.76279
	6	5	40.3	70.7	0.62392
	7	3	70.7	110.7	0.72696

Table 4.8. Downloaded bytes: k-means clustering for k=8–10.

k	Cluster number	Cluster size	Lower cluster boundary (MB)	Upper cluster boundary (MB)	Cluster silhouette coefficient
8	1	113	0.0	1.0	0.92901
	2	22	1.0	3.5	0.70751
	3	13	3.5	8.4	0.70863
	4	9	8.4	14.3	0.71754
	5	6	14.3	23.3	0.71955
	6	15	23.3	40.3	0.67502
	7	5	40.3	70.7	0.62392
	8	3	70.7	110.7	0.72696
9	1	81	0.0	0.3	0.92754
	2	35	0.3	1.2	0.58275
	3	19	1.2	3.5	0.64552
	4	13	3.5	8.4	0.68769
	5	9	8.4	14.3	0.71754
	6	6	14.3	23.3	0.71955
	7	15	23.3	40.3	0.67502
	8	5	40.3	70.7	0.62392
	9	3	70.7	110.7	0.72696
10	1	113	0.0	1.0	0.92901
	2	22	1.0	3.5	0.70751
	3	13	3.5	8.4	0.70863
	4	9	8.4	14.3	0.71754
	5	6	14.3	23.3	0.70299
	6	11	23.3	34.5	0.78432
	7	7	34.5	48.4	0.62645
	8	2	48.4	70.7	0.5498
	9	1	70.7	95.8	1.0000
	10	2	95.8	110.7	0.82208

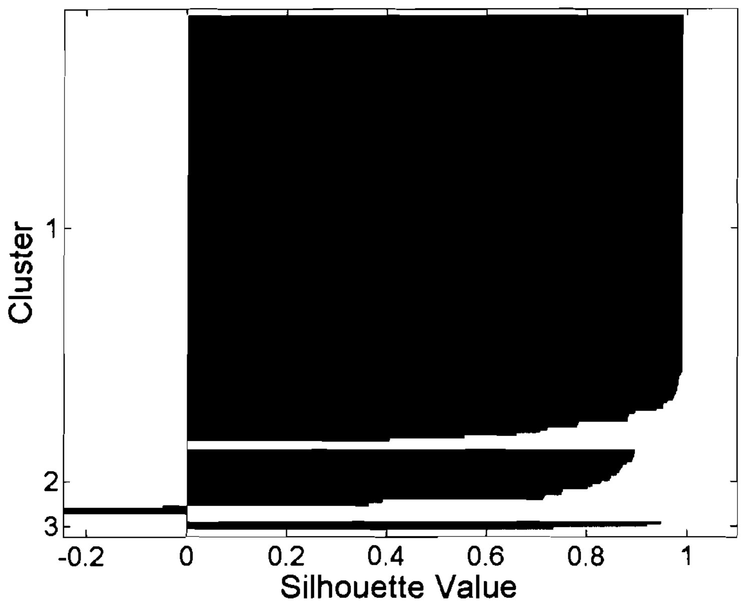

Figure 4.24. Silhouette plot: average downloaded bytes for k=3.

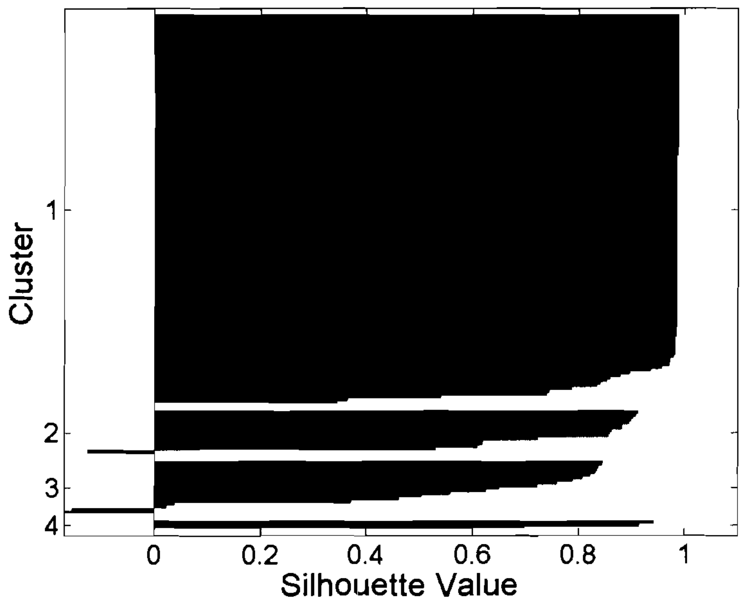

Figure 4.25. Silhouette plot: average downloaded bytes for k=4.

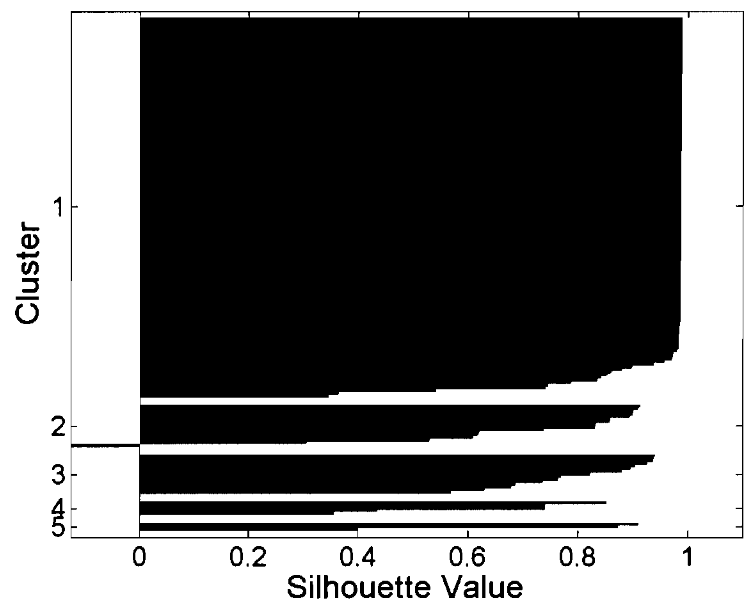

Figure 4.26. Silhouette plot: average downloaded bytes for k=5.

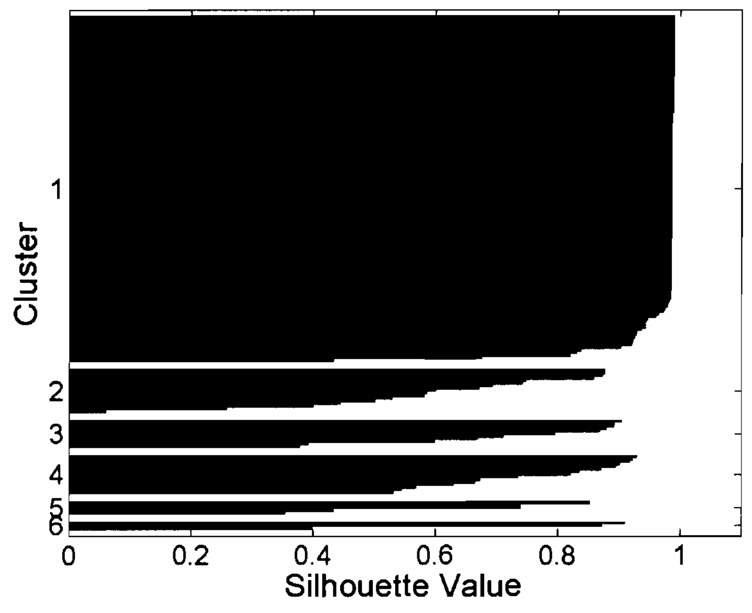

Figure 4.27. Silhouette plot: average downloaded bytes for k=6.

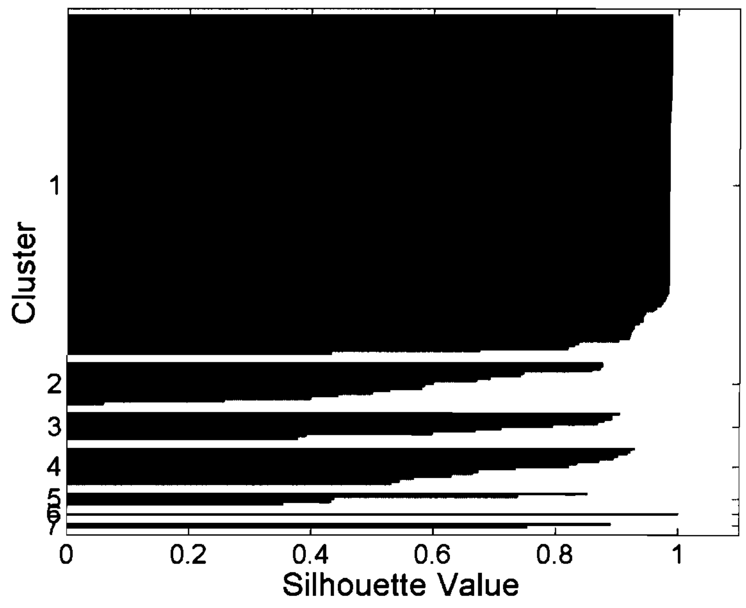

Figure 4.28. Silhouette plot: average downloaded bytes for k=7.

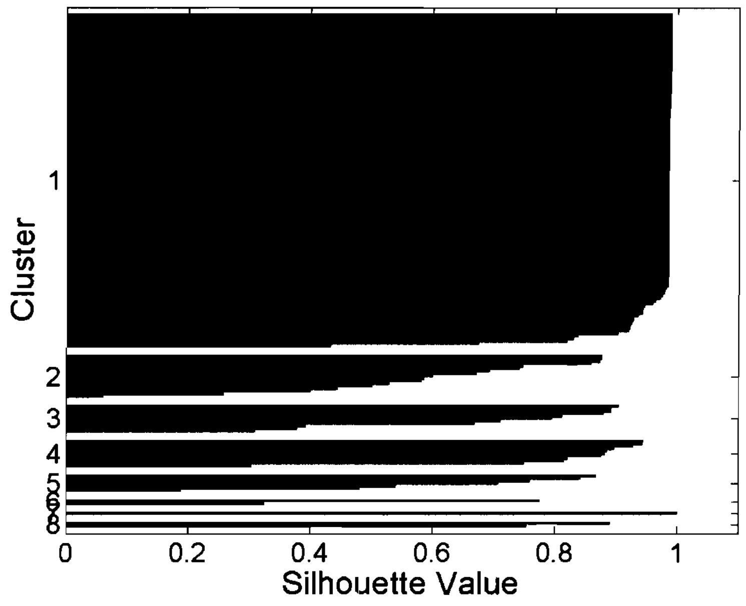

Figure 4.29. Silhouette plot: average downloaded bytes for k=8.

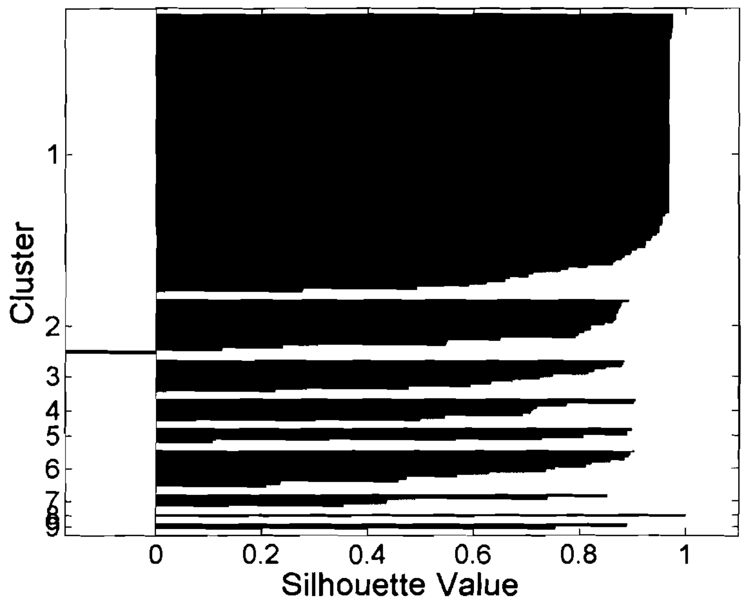

Figure 4.30. Silhouette plot: average downloaded bytes for k=9.

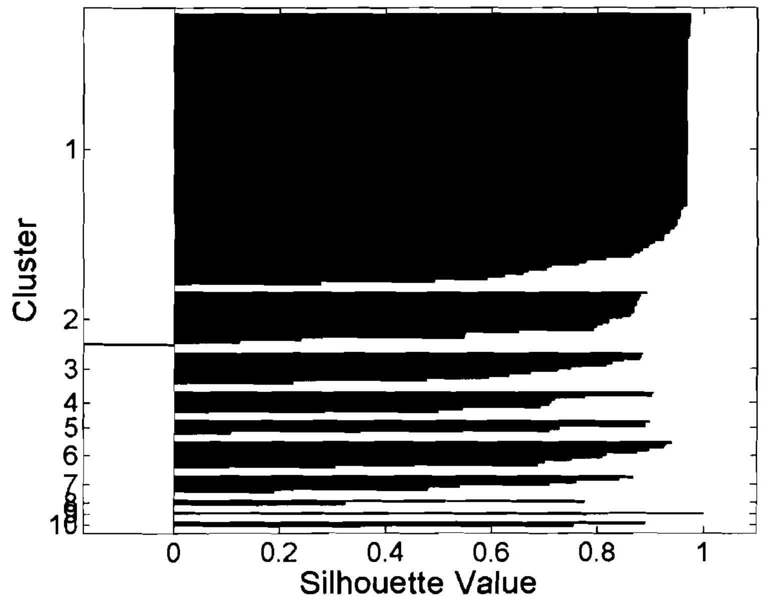

Figure 4.31. Silhouette plot: average downloaded bytes for k=10.

4.4.2.2 Multi-variable k-means clustering

For multi-variable k-means clustering, we combined the three variables from the Section 4.4.2.1: average downloaded bytes, average uploaded bytes, and average downloaded packets. On average, the number of downloaded and uploaded packets are identical. However, there is a relationship between the number of packets and number of bytes a user downloads/uploads. Due to the maximum packet size at the Ethernet layer, the number of packets downloaded/uploaded increases linearly as the number of bytes increases. Hence, we choose to employ average bytes per downloaded packet instead of average downloaded packets. These three variables are chosen because:

- Some users are web surfers and mainly download web pages. HTTP connections employ small request packets and large response packets. Hence, such users have high downloaded to uploaded bytes ratio. In contrast, users with interactive applications such as telnet, instant messaging, and Voice over IP may have a downloaded to uploaded bytes ratio closer to 1.
- TCP implementations and configurations may employ different MTU sizes. As described in Section 2.3, a larger MTU size improves TCP performance. Hence, users may be clustered based on their MTU size.

We employ the MATLAB k-means implementation in our analysis. Three dimensions (average downloaded bytes, average uploaded bytes, and average bytes per downloaded packet) and the *"cityblock"* distance measure were chosen. This distance measure sums the absolute differences in each dimension rather than measuring the Euclidean distance. The algorithm was repeated 15 times for each k from 2 to 9 and the clustering result was chosen with the MMSE, as defined in Section 3.1.1.1.

The results for the multi-variable k-means clustering is shown in Table 4.9 and Figure 4.32. The natural number of groups occurs at k=3. The results suggest that the separation between clusters is mainly due to downloaded and uploaded bytes.

Table 4.9. Average SC for multi-variables k-means clustering.

k	Average silhouette coefficient (multi-variables)
2	0.89438
3	0.92888
4	0.8949
5	0.91166
6	0.88658
7	0.87714
8	0.81643
9	0.73731

4.4.3 Classification of users by user activity

The k-means analysis from Section 4.4.2 grouped users by their traffic volume, which reduced the traffic for each user to a single value representing average packets and bytes. In this Section, we examine the traffic from each user by employing hierarchical clustering. We first classify the users by employing hierarchical clustering on their traffic patterns. We then refine the result of hierarchical clustering by classifying users based on the three most common traffic patterns.

Pattern matching of signals has been reported to be a difficult task [55], [56]. In the ChinaSat billing records, users have varying data traffic patterns. For example, two business users may have similar traffic patterns that do not occur on the same hour of the day (out of phase). Furthermore, bursty user traffic has different mean,

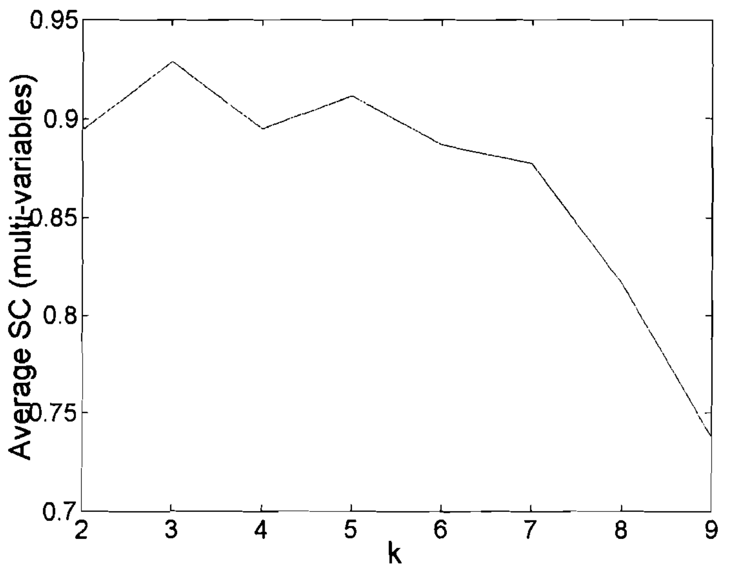

Figure 4.32. Plot of the average SC and k for multi-variable k-means clustering. The natural number of clusters for a set of objects correspond to the local maxima of the average SC.

peak, and variance. Hence, to simplify our analysis, the hourly traffic for each user is classified to values 1 (BUSY) or 0 (IDLE), as shown in Figure 4.33. For a particular hour, a user is considered BUSY if its billing record entry exists during the hour. Hence, BUSY indicates that a user has either downloaded or uploaded traffic during the particular hour. If no billing record exists for a user during an hour, the user is considered IDLE.

4.4.3.1 Hierarchical clustering

Not all users were BUSY for the entire recorded period. Hence, the similarity between two users' activities is calculated during the period when the users were BUSY instead of using the entire recorded period. We assigned a value of 1 for each hour that two users are either both BUSY or both IDLE and 0 otherwise. We call the sum

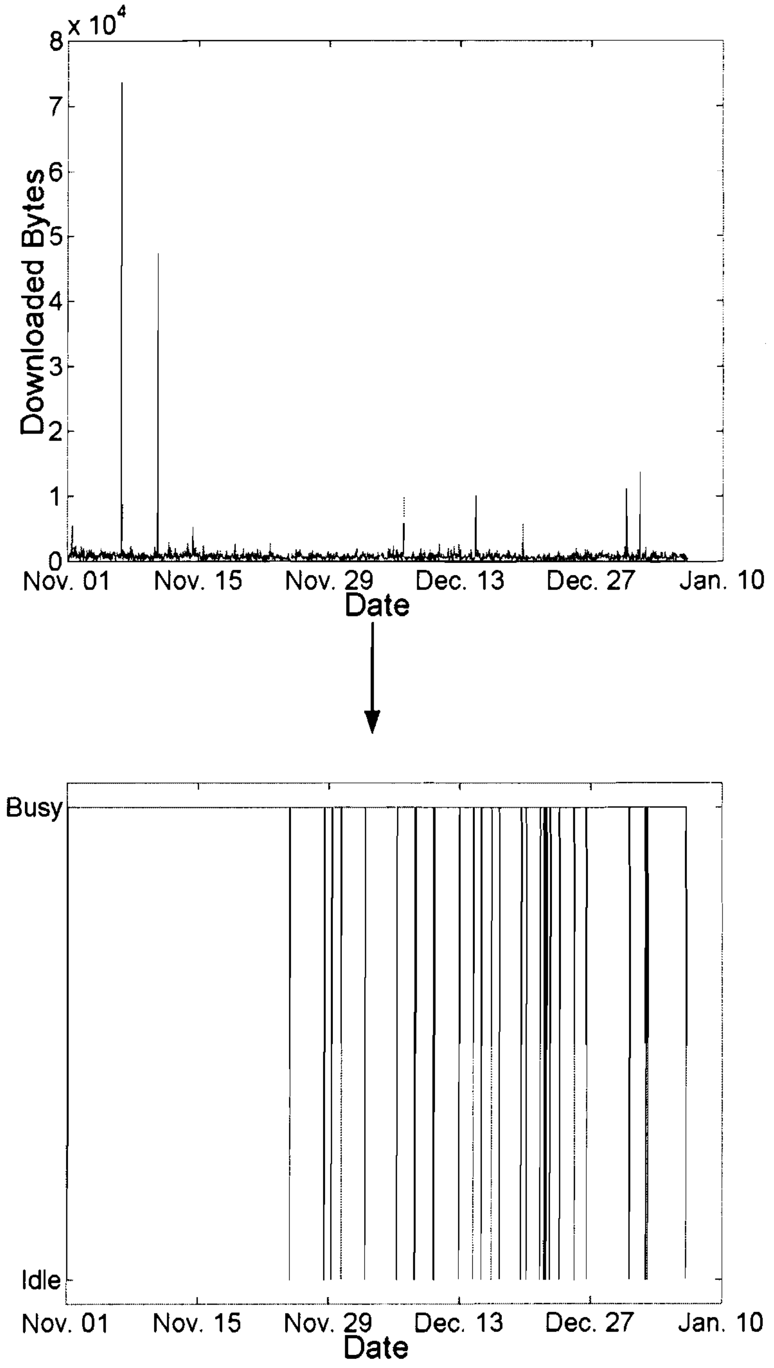

Figure 4.33. Classification of user traffic to values 1 (BUSY) or 0 (IDLE). For a particular hour, a user is considered BUSY if the user has either downloaded or uploaded traffic during the particular hour.

of the values the similarity score. The similarity score is normalized to the length of the recorded period. A similarity score of zero is assigned when the two traffic patterns do not overlap. Furthermore, some users may only be BUSY for a few hours in the billing records and have a short duration between their first and last day of activity. The shortness of the comparison period may result in high normalized similarity score between users who have a short activity duration and users who are mostly BUSY. However, we cannot remove users who have short activity duration from the analysis because transient users also contribute non-negligible traffic volume. Hence, to prevent users who are mostly IDLE from achieving a high similarity score with users who are mostly BUSY, we place a lower-bound on the minimum number of comparisons to be 3 weeks (504 hours). The similarity scores are placed into a similarity matrix of size 186×186.

The similarity matrix is then converted into a distance vector of size $n \times (n-1)$ (186×185) used by the MATLAB *linkage* function. For each of the four hierarchical trees constructed using the distance vector and the *linkage* function, we use a different distance measure. The four distance measures used are: *minimum distance*, *maximum distance*, *mean distance*, and *average distance*, as described in Section 3.1.2. Next, we calculate the cophenetic correlation coefficient (CPCC) for the four trees, as defined in Section 3.1.2.3. The correlation coefficients are shown in Table 4.10. Although the CPCC for the *average distance* measure is the highest, the clustering result is rejected because a non-monotonic tree is created, as shown in Figure 4.34. The non-monotonic links violate the hierarchical property of a tree. Hence, we choose the *mean distance* measure for hierarchical clustering.

The dendrogram plot for the 186 users using the means distance measure is shown

Table 4.10. Cophenetic correlation coefficients for the four distance measures. The average distance measure is rejected because the hierarchical tree is not monotonic. The mean distance is the best distance measure for creating a dendrogram of the traffic patterns.

Distance measure	Cophenetic correlation coefficient (CPCC)
Minimum distance	0.68900
Maximum distance	0.77610
Mean distance	0.92768
Average distance	0.93630

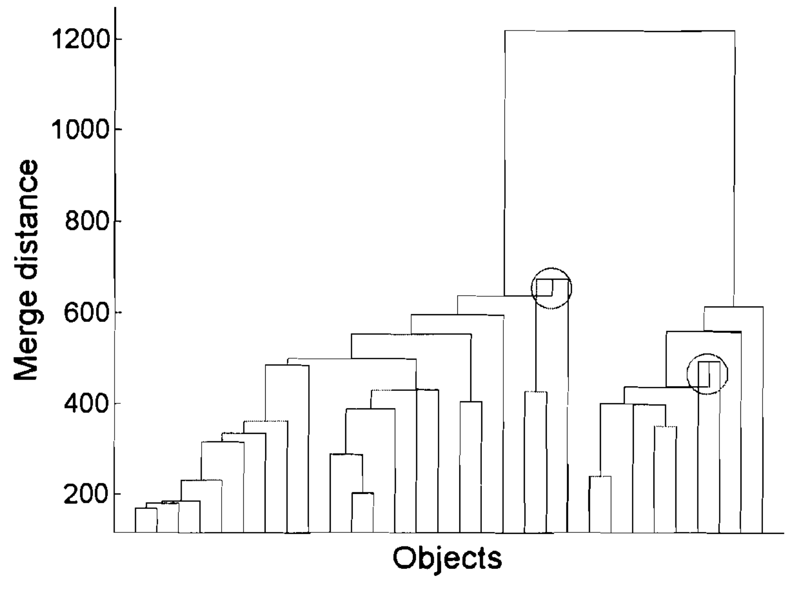

Figure 4.34. Dendrogram plot of the topmost 30 clusters by employing the average distance measure. The average distance measure is rejected because the created hierarchical tree is not monotonic. The two circled links that are not shaped as inverted "U" are not monotonic.

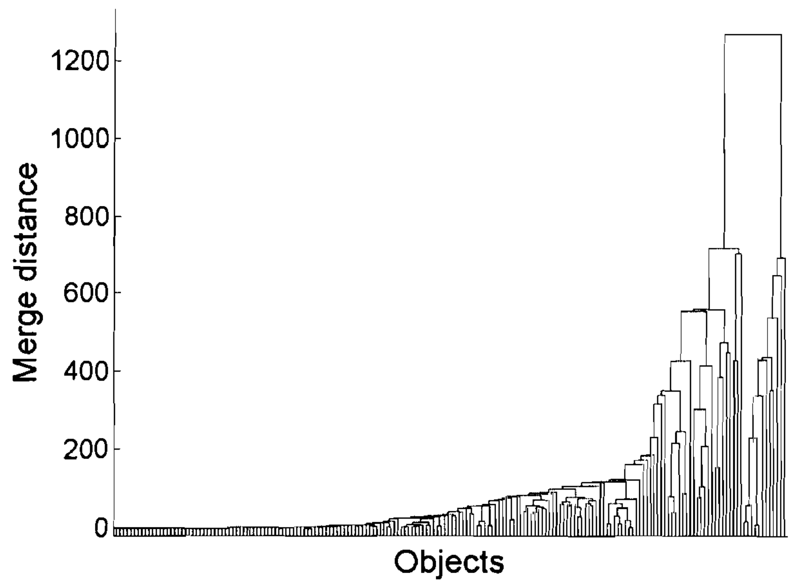

Figure 4.35. Dendrogram plot for all 186 users. The group of users clustered with small merged distances on the left side of the graph are mostly IDLE. The group of users on the right side with large merge distances exhibits cyclic activity.

in Figure 4.35. We employ the inconsistency coefficient described in Section 3.1.2.2 to find the number of clusters. The largest computed inconsistency coefficient is 1.1547. Hence, we select 1.10 (90% value) as cutoff for the inconsistency coefficient. This cutoff value results in 68 clusters. Setting the cutoff at 0.9 results in 76 clusters. This large number of clusters is caused by the comparison of traffic patterns of users whose activities do not overlap.

We choose k=3 to generate 3 clusters from the dendrogram. The results are shown in Table 4.11 and Figure 4.36. For clarity, only the topmost 30 clusters from Figure 4.35 are shown in Figure 4.36. By examining the user activity in each group, we found that group 1 contains users that are mostly IDLE for the duration of the recorded period and users that are BUSY 24 hours a day. No identifiable activity pattern can be found in group 2. Group 3 contains users who exhibit daily cycles of activity.

Table 4.11. Clustering results based on the hierarchical clustering and k=3. Group 1 contains users that are mostly IDLE for the duration of the recorded period and users that are BUSY 24 hours a day. No common user pattern can be found in group 2. Group 3 contains users who are BUSY 8-12 hours a day.

Group number	Number of users
1	171
2	3
3	12

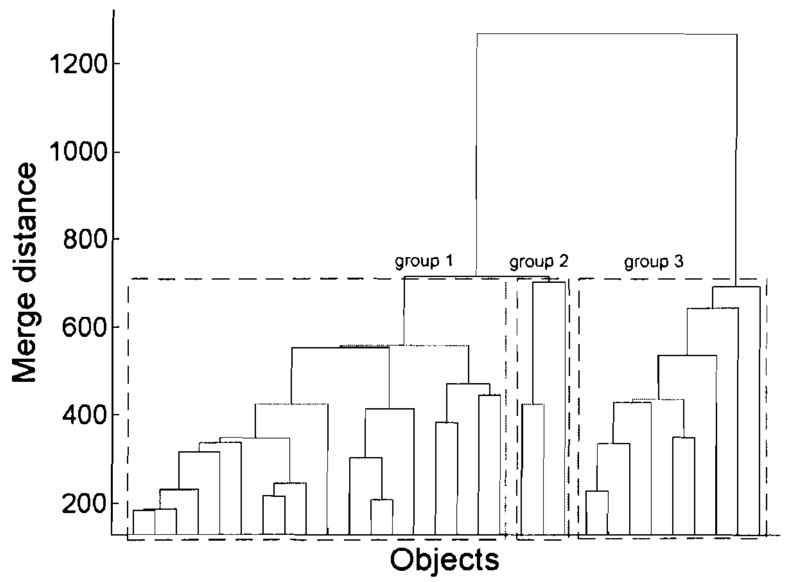

Figure 4.36. Dendrogram plot for the top 30 cluster tree nodes. Shown are the groups when k=3. Group 1 contains users that are mostly IDLE for the duration of the recorded period and users who are BUSY 24 hours a day. Group 2 has no identifiable pattern. Group 3 contains users who have cyclical activity patterns.

4.4.3.2 Clustering using the three most common traffic patterns

Using inconsistency coefficients, hierarchical clustering results in 68 clusters. From the largest clusters, we observed three most common traffic patterns, as shown in Figure 4.37. We assume that the traffic patterns of all users belong to one of the three patterns:

1. *Inactive* users: The first group of users are mostly IDLE during the recorded period. They are usually BUSY for less than 25% of the time and this group of users download/upload the least amount of data. Their behavior is approximated by a line of zero activity for the duration of the recorded period, as shown in Figure 4.38.

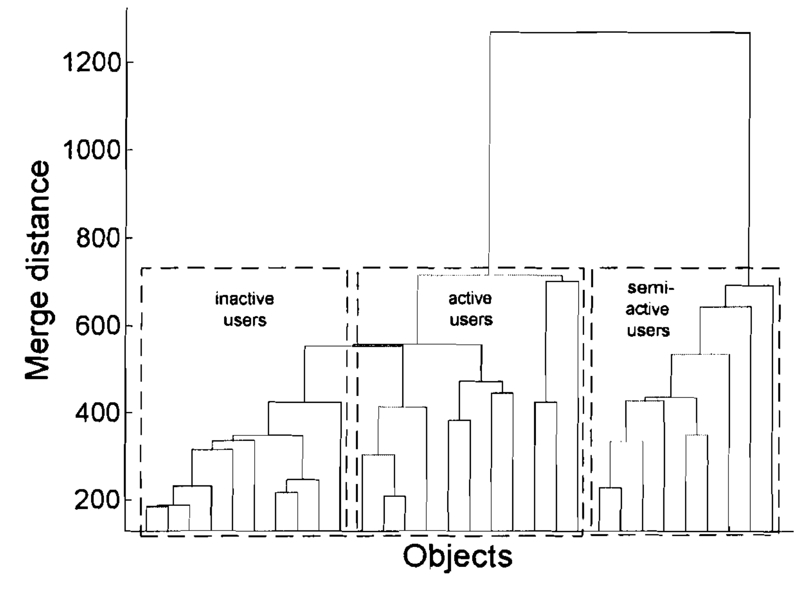

Figure 4.37. Dendrogram plot for the top 30 cluster tree nodes (3 most common traffic patterns). The leftmost group contains *inactive* users, the center group contains *active* users, and the rightmost group contains *semi-active* users.

2. *Active* users: The second group of users are BUSY for more than 18 hours a day. We conjecture that this group is comprised of users in 24-hour Internet cafés. We approximate their behavior by a line of full activity for the duration of the recorded period, as shown in Figure 4.39.
3. *Semi-active* users: The third group of users are BUSY for 8 to 12 hours a day. Although their BUSY hours overlap, their BUSY hours may not be identical. Their behavior is approximated by using a 10 hours BUSY/14 hours IDLE cycle for the duration of the recorded period, as shown in Figure 4.40.

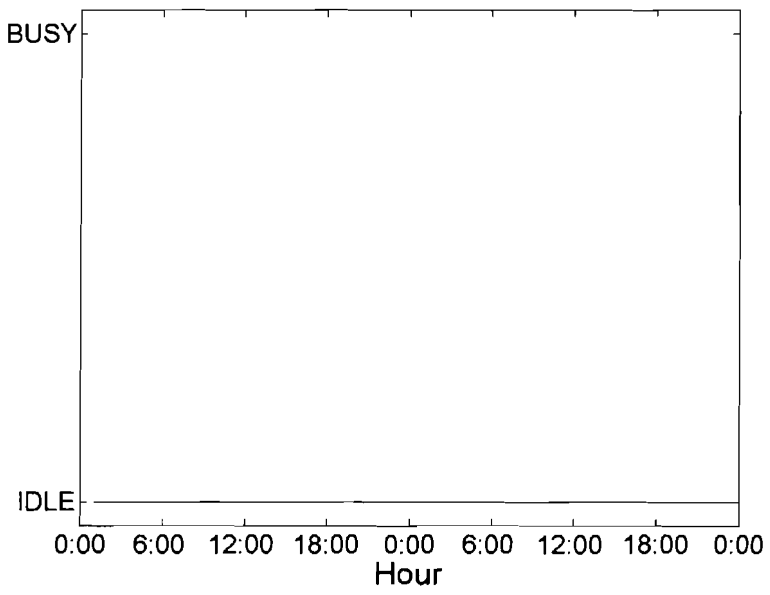

Figure 4.38. Traffic pattern of *inactive* users. Their behavior is approximated by a line of zero activity (IDLE hours) for the duration of the recorded period.

Users who are BUSY for 8-12 hours may be out of phase with the *semi-active* traffic pattern. To adjust for the phase variance, we translate the active pattern by a few hours so that the similarity score is maximized.

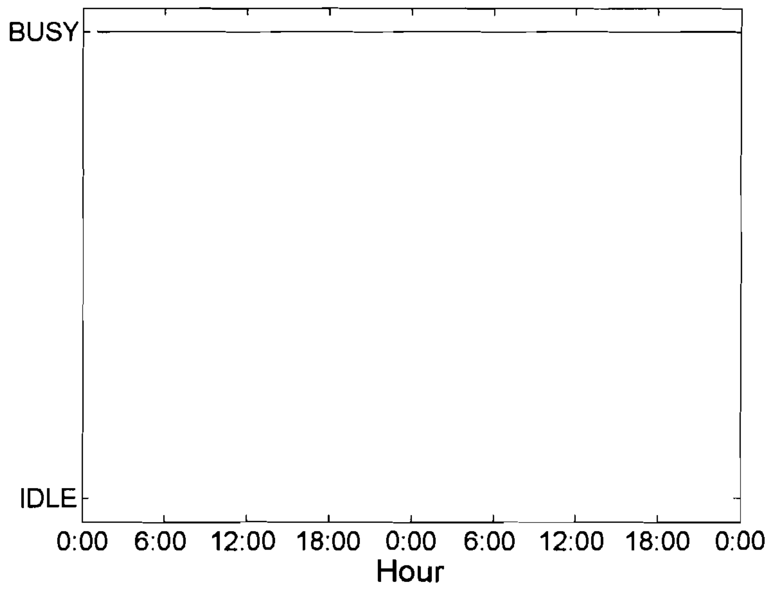

Figure 4.39. Traffic pattern of *active* users. Their behavior is approximated by a line of BUSY hours for the duration of the recorded period.

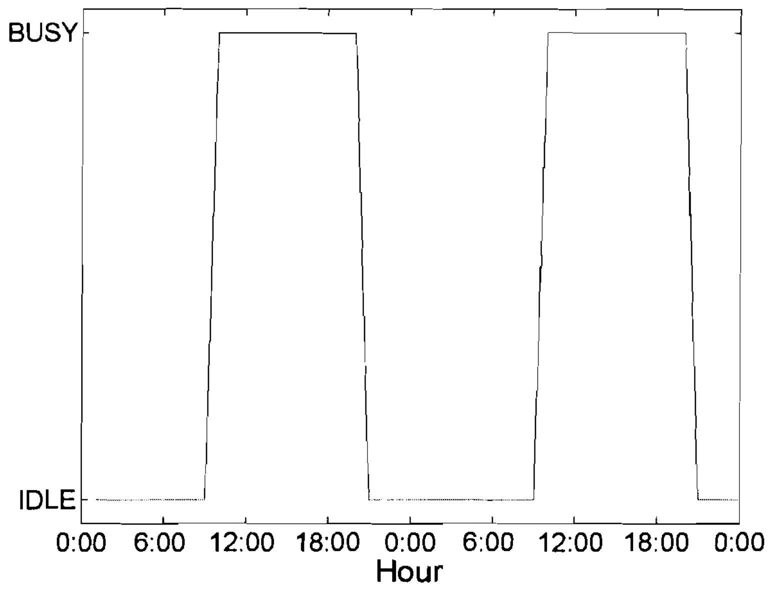

Figure 4.40. Traffic pattern of *semi-active* users. Their behavior is approximated by using a 10 hours BUSY/14 hours IDLE cycle for the duration of the recorded period.

We create a similarity matrix for each of the three patterns. Each user's traffic pattern is compared to the three common traffic patterns and a similarity score is recorded in the corresponding matrix. We group a user's traffic pattern into the cluster where the similarity score is the highest. The result of the clustering is shown in Table 4.12.

Table 4.12. Clustering results based on the three most common traffic patterns. Most users are labelled as *inactive* for the duration of the recorded period.

Traffic pattern	Number of users
Inactive users	162
Active users	16
Semi-active users	8

Although most users are classified correctly, some users may not fit the three chosen traffic patterns. For example, the user with *SiteID* 72805121 is identified as an *inactive* user even though the traffic pattern appears to be regular, as shown in Figure 4.41. The user was classified as *inactive* because the traffic does not exhibit a regular pattern. This user is usually *active* for 8 hours a day, but not during the same hours every day. There are also days during which this user was active for 12-15 hours. Thus, the user was classified as *inactive* even though the classification of *semi-active* may have been a better choice.

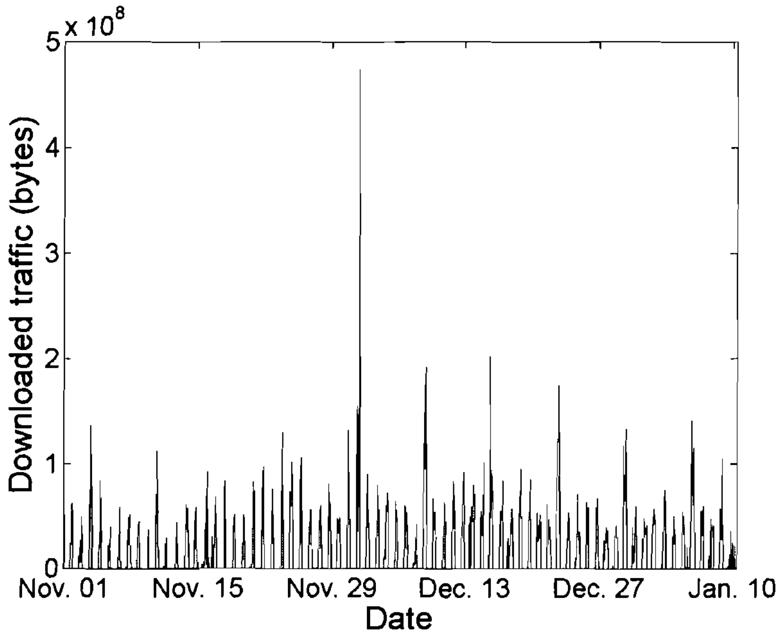

Figure 4.41. Downloaded bytes for user with *SiteID* 72805121. Although the traffic pattern appears regular, the user is not *active* on the same hours every day. This user is classified as *inactive* based on the three most common traffic patterns.

4.4.4 Combining clustering results from the k-means clustering and hierarchical clustering

We found no significant differences between using single and multi-variable clustering as described in Section 4.4.2, where k=3 was chosen as the natural number of clusters. Hence, we combine the single variable k-means clustering of downloaded bytes with hierarchical clustering. We use the three most common traffic patterns because hierarchical clustering produced too many clusters (using inconsistency coefficients) or clusters with no distinguishable patterns (choosing k=3). We employ the 3 most common traffic patterns and the best choice of k=3 from the k-means clustering. Hence, we categorize the users into 9 (3×3) clusters. However, only 8 clusters are present since one of the clusters has no objects. The 8 clusters have the following characteristics:

- Low traffic volume:
  1. *Inactive* users: This is the largest cluster with 150 members. The members concur with the histogram results given in Section 4.4.1, where 150 users downloaded less than 0.5×10^{10} bytes.
  2. *Active* users: 7.
  3. *Semi-active* users: 2.
- Medium traffic volume:
  1. *Inactive* users: 11.
  2. *Active* users: 9.
  3. *Semi-active* users: 4.
- High traffic volume:
  1. *Inactive* users: Only one user belongs to this cluster. This particular user contributed a large amount of traffic while BUSY, as shown in Figure 4.42. However, the user was only active between Dec. 24, 2002 and Jan. 10, 2003 and generated no regular traffic pattern.
  2. *Semi-active* users: The two users belonging to this cluster are most likely businesses or Internet cafés. They contributed the largest volume of total traffic and are open for $\sim$10 hours a day.

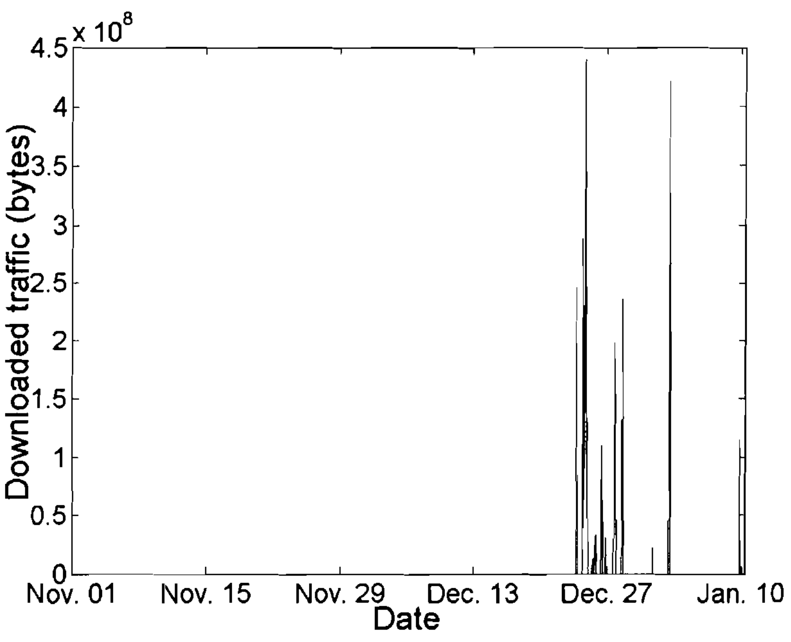

Figure 4.42. Average traffic volume (downloaded bytes) for user with *SiteID* 72640513. The user was *active* between Dec. 24, 2002 and Jan. 10, 2003. The user contributed a large volume of traffic but generated no regular traffic pattern.

Chapter 5

Analysis of *tcpdump* traces

The PEP techniques employed in the DirecPC network reroute all satellite user traffic to the NOC, as described in Section 2.3.5. Hence, the NOC is the ideal location to collect traffic traces. The traces were collected from a port on the primary Cisco router at the NOC, located in the Northwest rural area of Beijing, China. The router provides access to the inbound and outbound packets sent between the hosts using the NOCs 100 Mbps local area network (LAN). The NOC connects to the Internet backbone through a 10 Mbps link.

We employed the open-source passive network monitor tool *tcpdump* to collect the traffic traces. The tool was installed on a Linux PC equipped with a 100 Base-T Ethernet adaptor and a high-resolution (100 μs) timer. The *tcpdump* tool was configured to capture the first 68 bytes of each packet to ensure user privacy and to minimize storage requirements while preserving the IP and TCP headers. The TCP payload was not collected. The *tcpdump* traffic traces were continuously collected from 11:30 on Dec. 14, 2002 to 11:00 on Jan. 10, 2003. The collected traces were stored in 127 files, containing ~63 GB of data.

5.1 *pcap* file format

The *tcpdump* tool stores packets using the interface provided by the packet capture library *libpcap*. Each packet trace (*pcap* file) contains a header section and a data section. The layout of a *pcap* file is shown in Figure 5.1.

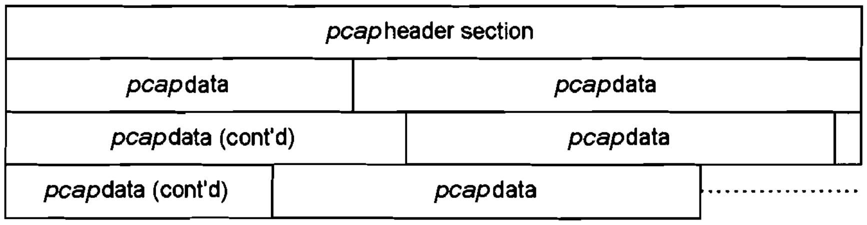

Figure 5.1. General layout of a *pcap* file. Each *pcap* file contains a header section that describes the parameters used by *tcpdump*. *pcap* data include timestamps, packet lengths, and the captured packet. They are variable in length.

The header fields of a *pcap* file and their sizes are shown in Figure 5.2. The first field, *magic number*, contains a hexadecimal value of 0xa1b2c3d4 for big-endian systems or 0xd4c3b2a1 for little-endian systems. The endianness of a system determines the storage order multi-byte data. Big-endian systems store the most significant byte (MSB) at the lowest memory address while little-endian systems store the lowest significant byte (LSB) at the lowest memory address. This magic number specifies whether the multi-byte fields should be read in the big-endian or little-endian order. The fields in Figures 5.2 - 5.4 labelled with "*" are impacted by the endianness of a system. *Pcap major version* and *Pcap minor version* describe the version (*major.minor*) of *libpcap* employed to record the trace. All ChinaSat *tcpdump* traces have

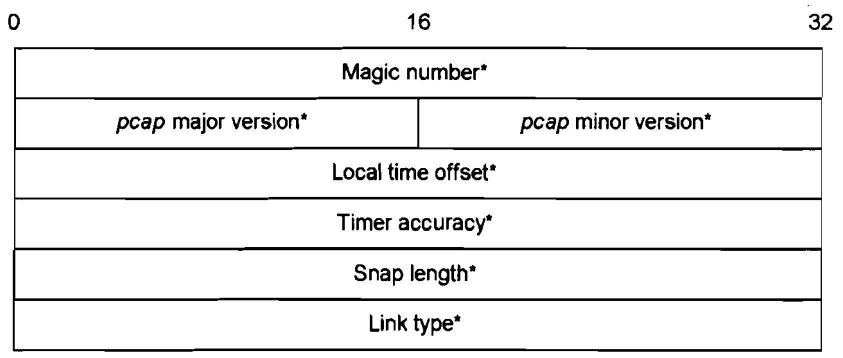

Figure 5.2. The fields of a *pcap* file header. Magic number, local time offset, timer accuracy, snap length, and link type are all 32 bits in length. *pcap* major version and *pcap* minor version are 16 bits in length. All fields labelled with "*" are affected by the endian order.

been recorded using *libpcap* version 2.4. *Local time offset* value indicates the difference (seconds) between the coordinated universal time (UTC) and the local time if the recording machine uses UTC. The *timer accuracy* value indicates the precision of the timer (microseconds). In the recorded *tcpdump* traces, both local time offset and timer accuracy values use the default value of zero, implying that the timestamps employ local time. *Timer accuracy* value zero means that the timer precision is not specified. *Snap length* indicates the maximum number of bytes that will be captured from each packet. In the recorded *tcpdump* traces, *snap length* has the default value of 68 bytes. Hence, only the first 68 bytes of a packet are captured if the packet size is larger than 68 bytes. Packets with sizes smaller than 68 bytes are fully captured. *Link type* indicates the data link layer protocol used by the recording device. For the recorded *tcpdump* trace, the link employed was Ethernet with *link type* EN10MB (value = 1). EN10MB specifies Ethernet link speeds 10Mb/s, 100Mb/s, and 1,000Mb/s. The *pcap* header values from the ChinaSat *tcpdump* traces are shown in Table 5.1.

Table 5.1. Default header field values of a *pcap* file header.

Pcap file header field name	Field value
Magic number	0xd4c3b2a1 (little endian)
Pcap major version	0x02
Pcap minor version	0x04
Local time offset	0 (not used)
Timer accuracy	0 (not used)
Snap length	0x44 (68 bytes)
Link type	1 (EN10MB: 10/100/1,000Mb/s Ethernet)

The data section of a *pcap* file contains numerous entries with packet-related information such as timestamps, packet lengths, and the captured packet, as shown in Figure 5.3. There are two timestamp fields: *seconds* and *microseconds.* The *seconds* field employs the Unix time format (the number of seconds elapsed since midnight on the morning of Jan. 1, 1970). The *seconds* field stores either the UTC or the local time depending on the value of local time offset field. The *microseconds* field records the number of microseconds that has elapsed during the recorded second. There are two fields for packet length: *recorded packet length* and *actual packet length.* The *recorded packet length* field records the minimum of either the snap length (68 in our recorded *tcpdump* traces) or the actual packet size. The *actual packet length* field records the total length of the packet. The captured packet field includes the link layer frame and the IP datagram and is preceded by timestamp and packet length fields.

The link layer header (Ethernet header) from the ChinaSat *tcpdump* trace is shown in Figure 5.4. It contains destination and source Ethernet addresses (6 bytes each) and the Ethernet (frame) type (2 bytes). In the ChinaSat *tcpdump* trace, only three

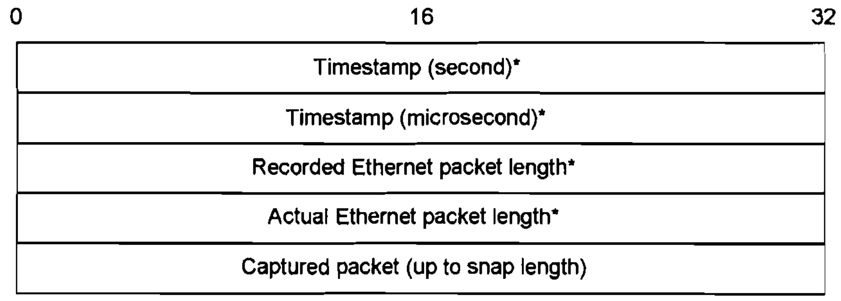
0
16
32
Timestamp (second)*
Timestamp (microsecond)*
Recorded Ethernet packet length*
Actual Ethernet packet length*
Captured packet (up to snap length)

Figure 5.3. The fields for each *pcap* data entry. The two timestamp and the packet length fields are all 32 bits in length. The captured packet can be as long as the *snap length*, which is 68 bytes.

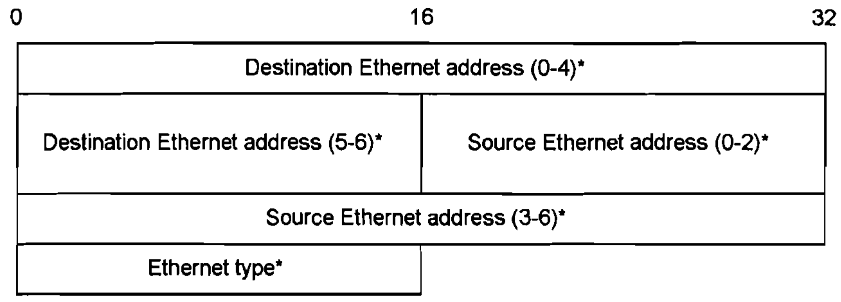
0
16
32
Destination Ethernet address (0-4)*
Destination Ethernet address (5-6)*
Source Ethernet address (0-2)*
Source Ethernet address (3-6)*
Ethernet type*

Figure 5.4. The fields of an Ethernet header. Ethernet addresses are 48 bits long. The 16-bit Ethernet type field indicates the type of a link.

distinct Ethernet addresses are recorded. One of the three Ethernet addresses belongs to the router. The other two addresses belong to the next-hop routers where data from the Internet and from the ChinaSat users are sent/received, respectively. The Ethernet type value recorded is 0x0800, corresponding to the value for Internet Protocol version 4 (IPv4).

The IP datagram is preceded by the Ethernet header and is not affected by the endian order. The transport layer packets (ICMP, UDP, and TCP) follow the IP header and are shown in Figures 5.5 – 5.7.

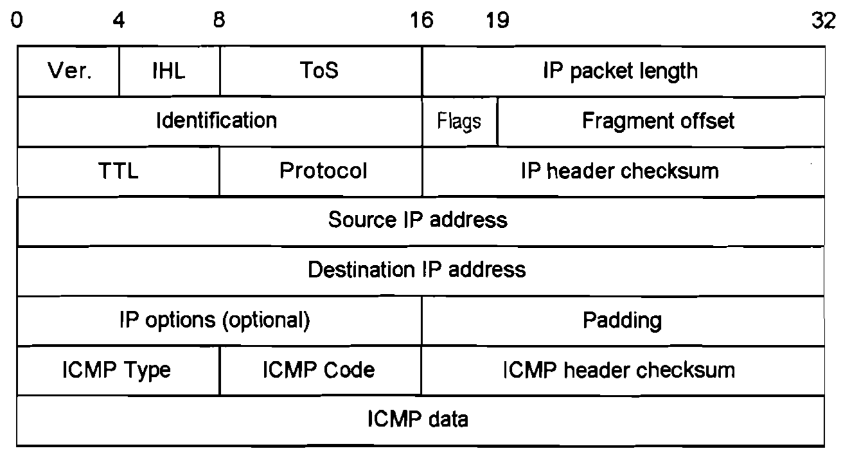

Figure 5.5. ICMP packet format.

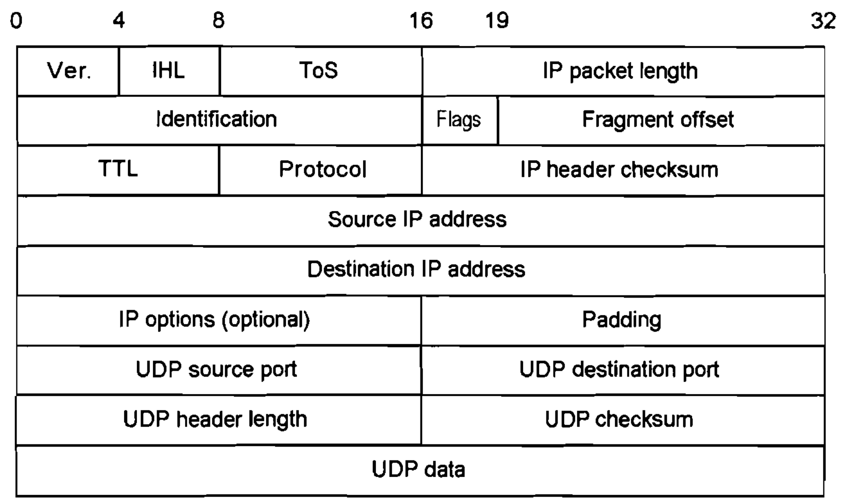

Figure 5.6. UDP packet format.

The IP header is common to all three segments [57], [58]. The fields include the IP version number, IP header length (IHL), type of service (ToS), total packet

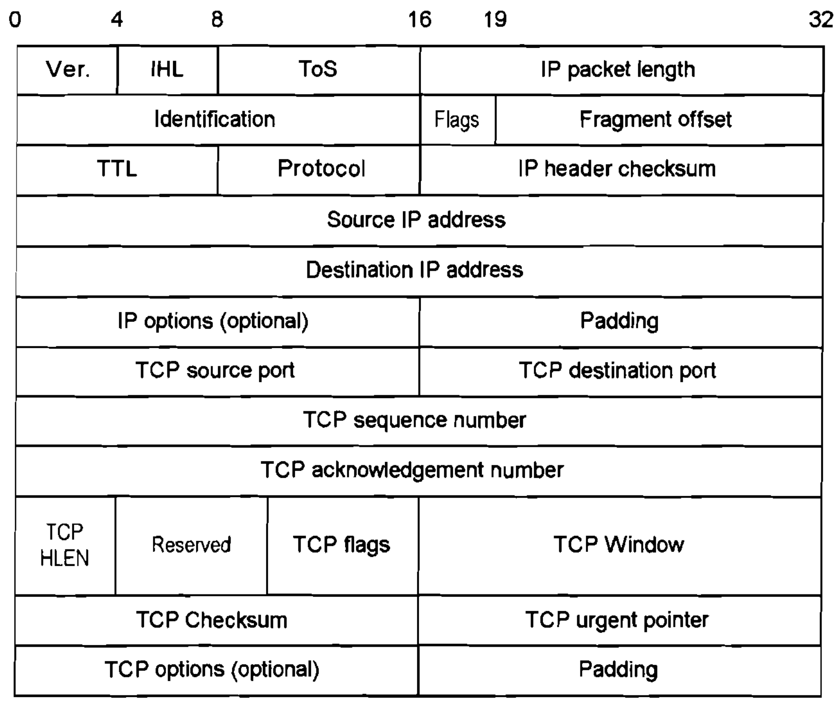

Figure 5.7. TCP packet format.

length, identification, IP flags, fragment offset, time-to-live (TTL), transport-layer protocol type, IP header checksum, source and destination IP addresses, optional IP options, and padding. In the ChinaSat *tcpdump* trace, the IP version number used was 4 because IPv6 was not widely deployed in 2002. The IP header length field indicates the length of the IP header measured in 32-bit words. The ToS field is not used in the ChinaSat network. The total length field records the total size of the datagram measured in bytes, including the IP header and the IP data. This value is used for analysis. The identification field contains a unique integer that identifies each IP datagram. The 3-bit flags field contains three boolean values from highest bit to lowest: "reserved", "do not fragment", and "more fragments". A set "do not fragment" bit is a signal to intermediate routers not to fragment the packet. An ICMP error message should be returned if the packet cannot be transmitted in its entirety. The "more fragments" bit indicates that a datagram is a fragment from a larger datagram. All datagram fragments have the same identification value and all fragments except for the final one have the "more fragment" bit set. The fragment offset field specifies a datagram fragment's location in the original datagram The TTL field indicates the remaining number of hops a datagram is allowed to traverse. Each intermediate router has to reduce the TTL value by 1 before forwarding an IP datagram. When the TTL value is zero, the IP datagram is discarded and an ICMP error message is returned to the sender. Various TCP/IP implementations employ different default TTL values. The protocol field indicates the transport layer protocol. For the recorded *tcpdump* trace, segments from three transport protocols are captured: ICMP (protocol value 1), UDP (protocol value 17), and TCP (protocol value 6). IP checksum ensures the integrity of the header values. The source and destination IP address fields contain the source and destination IP addresses. In the recorded trace, the ChinaSat network users employ IP addresses in the range 192.168.1.1 - 192.168.2.255. This address range is part of the private IP address space [59]. The use of private IP addresses in a deployed network indicates that Network Address Translation (NAT) [60] and dynamic IP [61] are employed. The IP options field contains additional IP options, if they are used. None were recorded in the ChinaSat *tcpdump* traces. Lastly, if an IP datagram does not end on the 32-bit word boundary due to IP options, a variable length padding value of zero is added to fill the remaining bits.

In addition to the IP header portion, an ICMP packet contains 4 fields: ICMP type, ICMP code, ICMP header checksum, and ICMP data. In the ChinaSat network, we only detect two ICMP types: *echo request* (type 8) and *echo reply* (type 0). The code field is zero for both types. ICMP header checksum ensures ICMP header integrity. ICMP data is used for padding and specifying the size of a *echo request.* From the UDP header, we are interested in the value of the destination port, which may identify common applications. TCP header has several fields of interest: TCP destination port, TCP flags, and TCP options. The TCP destination port identifies the application, as shown in Table 5.2. The TCP flags are used for connection establishment and termination. The TCP options field specifies extensions to the original TCP protocol [23] and are employed to enhance performance.

Table 5.2. Common TCP applications sorted by ports used.

TCP application	Full name	TCP Port
FTP data	File transfer protocol data	20
FTP control/command	File transfer protocol control/command	21
SSH	Secure shell protocol	22
Telnet	Teletype network protocol	23
SMTP	Simple mail transfer protocol	25
HTTP	Hypertext transfer protocol	80
POP3	Post office protocol version 3	110
NETBEUI	NetBIOS extended user inferface	139
IRC	Internet relay chat	194
HTTPS	HTTP over secure socket layer (SSL)	443
MS SQL	Microsoft structured query language server	1433

5.2 Constancy of IP addresses

The ChinaSat users have allocated private IP addresses in the range of 192.168.1.1 – 192.168.2.255. The use of a private IP address range is an indication that NAT [60] and dynamic IPs [61] are used. When these two techniques are deployed, a user's IP may change every time the computer connects to the network. While we may assume that a satellite user retains the same IP over a few hours, a user may not retain the same IP over a few days. Hence, a particular IP address could belong to two different satellite users at separate times. User analysis for the full duration of the three weeks trace cannot be performed because it is not possible to identify a particular satellite user in the *tcpdump* traces. As a consequence, cluster analysis cannot be performed. We also cannot associate satellite user *SiteIDs* with IP addresses to gain additional insights. Instead, we analyze the behavior of users by assuming that the IP addresses remain constant for a few hours.

5.3 General characteristics of traffic data

5.3.1 Protocols and applications

It is not surprising that the collected traffic traces contain only IP packets because IP is the most widely used network layer protocol. We did not capture traffic from protocols such as the address resolution protocol (ARP) [62] and the reverse address resolution protocol (RARP) [63] due to the *tcpdump* defaults. The distribution of traffic data by protocols is shown in Table 5.3. We also analyze the activity by TCP port numbers because TCP accounts for majority of the packets. Traffic data in terms of applications, connections, and bytes are shown in Table 5.4. World Wide Web (WWW) traffic (port 80) is the most widely used TCP application in terms of number of bytes, followed by FTP. Approximately 10% of all connections use unknown ports.

Table 5.3. Characteristics of traffic data sorted by protocols.

Protocol	Bytes (%)	Packets (%)
TCP	94.50	84.30
UDP	5.06	14.20
ICMP	0.45	1.45
Total	100.00	100.00

Table 5.4. Characteristics of traffic data sorted by TCP applications.

Applications	Connections (%)	Bytes (%)
WWW (80)	90.00	76.800
FTP-data (20)	0.20	10.700
IRC (194)	0.80	0.008
SMTP (25)	0.10	0.010
POP3 (110)	0.03	0.020
Telnet (23)	0.02	0.002
Others	8.90	12.500
Total	100.00	100.00

Only a few known applications use a standard UDP port. UDP, an unreliable transport layer protocol, is mainly used for real-time applications such as video streaming and Internet telephony. Many of these applications use random ports. Hence, we cannot identify the majority of UDP applications based on UDP ports and can only identify the Routing Information Protocol (RIP) [64] packets transmitted on UDP port 520. RIP is used for packet routing between various hosts in a local network. The RIP packets were sent between the three Ethernet addresses described in Section 5.1. Although we are able to identify a large number of RIP packets, they are not related to the DirecPC traffic in the ChinaSat network. Therefore, we did not analyze these packets further.

5.3.2 TCP options

In Section 2.3, we described TCP options such as SACK, the sliding window scale option, increasing the initial *cwnd*, and path MTU discovery. These extensions are requested during the TCP three-way handshake. Hence, we examine the initial two segments (SYN and SYN/ACK) of the TCP connections and identify that SACK is widely used in the ChinaSat network. Over 60% of connections support the SACK option. Less than 5% of connections use the sliding window scale option. The commonly deployed Microsoft Windows OS versions 98 and higher support and enable SACK by default [65]. The sliding window scale option is disabled by default. A small number of Linux distributions employ sliding window scale option with the value of zero. Hence, the prevalent usage of SACK and the infrequent usage of sliding window scale option in the recorded *tcpdump* traces are caused by the Microsoft Windows TCP implementation. In addition, most connections use the window size of 4 MSS or larger. This is also the default for Windows. Lastly, there were no instances of path MTU discovery. Most TCP implementations use a default MSS size of 1,460 bytes instead of searching for a maximum MTU.

5.3.3 Operating system fingerprinting

TCP SYN packets can be examined to identify the end users' operating systems (OSes) through techniques called OS fingerprinting. These techniques and are used for intrusion detection, vulnerability discovery, and network auditing. In this Section, we show that Microsoft Windows is the cause of the observed TCP options in the ChinaSat network.

OS fingerprinting techniques are based on the fact that TCP/IP implementations are unique [66]–[69]. For example, captured packets show that Microsoft Windows enable SACK and set MSS to 1,460 bytes by default in TCP SYN packets [70]. Since TCP options end on the 32-bit boundary, two TCP no operation (NOP) options are used for padding. However, the Windows implementation is unique because the NOPs are placed in front of SACKOK in the following order: MSS, NOP, NOP, and SACKOK.

In addition to the order of TCP options, the IP TTL value, the TCP window size, the IP DF flag, the IP ToS bits, and the TCP SYN packet size are also used to identify an OS [67], [69]. The signatures for a few common OSes are listed in Table 5.5.

The TCP/IP implementation of different OSes can be determined actively or passively. Active OS fingerprinting techniques send SYN probes with various TCP options to hosts and determines the hosts' OSes based on the replies. In contrast, passive fingerprinting determines the OSes based on captured packets. We use the passive open source OS fingerprinting tool *p0f v2* [68], which supports the *pcap* file format.

For this OS fingerprint analysis, we choose the *tcpdump* traces collected over the period of 9 hours on Dec. 14 and assume that the user IPs is constant throughout.

Table 5.5. TCP SYN defaults for common Operating Systems.

OS name	TTL	Window size	DF	ToS	Packet size	TCP options
Microsoft Windows	128	16384	Y	0	48	MSS, SACKOK, 2 NOPs
IBM AIX	64	16384	Y	0	44	MSS
FreeBSD	64		Y	16	64	MSS, SACKOK
OpenBSD	64	16384	N	16	64	MSS, SACKOK, WSCALE, 5 NOPs
Linux	64	5840	Y	0	60	MSS, SACKOK, WSCALE, 1 NOP

The results from the analysis are shown in Table 5.6. We detected 171 users, of which 137 are *inactive*. *Inactive* users did not initiate TCP connections and, thus, we are not able to determine their OS. Of the 17 *active* users, fourteen use Microsoft Windows and two use Linux. The *p0f* tool identifies the unknown OS to be a MSS modifying proxy. Even though the OS of the *inactive* users cannot be identified, the distribution of *active* users suggest that the majority of ChinaSat users rely on Microsoft Windows OS.

5.4 Data traffic anomalies

We use open-source programs *Ethereal/Wireshark* [71], *tcptrace* [72], and the developed program *pcapread* to examine the traffic traces. Analysis of the *tcpdump* traces reveals data traffic anomalies such as packets with invalid TCP flag combinations, large number of connections closed using TCP reset, port scans, and traffic volume anomalies.

Table 5.6. OS fingerprinting results. A total of 171 users are detected, of which 17 are *active*. 14 of the *active* users employ Microsoft Windows, 2 users employ Linux as their OS, and 1 user employ OS that could not be determined.

User type *active/inactive*	Operating system	Number of users
inactive		137
active		17
	Microsoft Windows	14
	Linux	2
	Unknown	1
Total		171

5.4.1 Packets with invalid TCP flag combinations

TCP SYN, FIN, and RST flags are used to open connections, close connections regularly, and close connections when an error occurs, respectively [23]. The TCP PSH flag allows a TCP application to transmit all outstanding packets in the buffer without delay. Packets with more than one SYN/FIN/RST flag set are invalid. Furthermore, the TCP PSH flag cannot be used in combination with RST. Invalid flag combinations may cause TCP/IP implementations to exhibit unexpected behavior or fail. They are also used to test TCP/IP robustness [73]. Hence, it is unusual to find packets with combinations of the TCP flags. Packets with invalid combinations may be sent by malicious programs, viruses, or worms. A vulnerable TCP/IP implementation may exhibit unexpected behavior even with a single invalid packet. The number of discovered packets with invalid TCP flag combinations is shown in Table 5.7. 0.3% of packets with TCP open/close flags have invalid combinations.

Table 5.7. Packets with various TCP flag combinations. Marked with "*" are invalid TCP flag combinations.

TCP flag	Packet count	% of Total
SYN only	19,050,849	48.500
RST only	7,440,418	18.900
FIN only	12,679,619	32.300
**SYN+FIN*	408	0.001
**RST+FIN (no PSH)*	85,571	0.200
**RST+PSH (no FIN)*	18,111	0.050
**RST+FIN+PSH*	8,329	0.020
**Total number of packets with invalid TCP flag combinations*	112,419	0.300
Total packet count	39,283,305	100.000

5.4.2 Large number of TCP resets

A TCP connection is opened with the SYN flag and closed with the FIN flag. However, data shown in Table 5.7 indicate that 37% (7,440,418 / (7,440,418 + 12,679,619)) of connections are closed by the RST flag. This is caused by Microsoft Internet Explorer that employs RST instead of FIN to close connections in order to improve web browsing performance [74]. This concurs with results reported in Section 5.3.3, where most ChinaSat users are found to employ Microsoft Windows OS.

5.4.3 Port scans

Port scans are usually malicious in intent. In the ChinaSat network, both UDP and TCP port scans are present.

5.4.3.1 UDP port scans

Analysis of the *tcpdump* traces shows that UDP port scans occur on port 137, both originate from and are directed to the ChinaSat network. UDP port 137 is used by the Microsoft NETBEUI (NETBIOS extended user interface) protocol, which enables file and printer sharing in a local network of Windows PCs. NETBEUI usually employs UDP port 137 at both endpoints. Hence, traffic from UDP port 137 to other UDP ports or traffic from other UDP ports to UDP port 137 indicate abnormal behavior.

An example of a host in the ChinaSat network (IP address 192.168.2.30) that transmitted packets to Internet hosts from UDP port 137 is shown in Table 5.8. For a certain destination IP (202.y.y.226), the ChinaSat host transmitted packets to multiple ports (1025, 1027, 1028, and 1029). This behavior is known as a port scan and usually indicates malicious intent. An example of a host external to the ChinaSat network (210.x.x.23) that transmitted packets from UDP port 1035 to ChinaSat hosts at the destination UDP port 137 is shown in Table 5.9. Two Internet worms, Bugbear and Opasoft, were prevalent when the *tcpdump* traces were captured. Both worms use the NETBEUI protocol to propagate to other hosts. Without having the UDP payload recorded, we are unable to determine if these two worms indeed generated the port scans.

5.4.3.2 TCP port scans

TCP port scans on TCP ports 80, 139, 443, 1433, and 27374 are detected in the *tcpdump* traces. The detected scans were directed to the ChinaSat users. TCP port 139 is the TCP NETBEUI port. Similar to the port scans on UDP port 137, these packets are malicious in intent. On Dec. 14, 2002, three external addresses were

Table 5.8. Port scan originating from the ChinaSat network. Targets are scanned at random from UDP port 137. For some destinations (202.y.y.226), multiple UDP ports (1025, 1027, 1028, and 1029) are scanned.

Origin IP:port	Destination IP:port
192.168.2.30:137	195.x.x.98:1025
192.168.2.30:137	202.x.x.153:1027
192.168.2.30:137	210.x.x.23:1035
192.168.2.30:137	195.x.x.42:1026
192.168.2.30:137	202.y.y.226:1026
192.168.2.30:137	218.x.x.238:1025
192.168.2.30:137	202.y.y.226:1025
192.168.2.30:137	202.y.y.226:1027
192.168.2.30:137	202.y.y.226:1028
192.168.2.30:137	202.y.y.226:1029
192.168.2.30:137	202.y.y.242:1026

Table 5.9. Port scan directed to the ChinaSat network. The Internet host (210.x.x.23) sent packets from UDP port 1035 to the Microsoft NETBEUI port (137) at multiple ChinaSat network hosts.

Origin IP:port	Destination IP:port
210.x.x.23:1035	192.168.1.121:137
210.x.x.23:1035	192.168.1.63:137
210.x.x.23:1035	192.168.2.11:137
210.x.x.23:1035	192.168.1.250:137
210.x.x.23:1035	192.168.1.25:137
210.x.x.23:1035	192.168.2.79:137
210.x.x.23:1035	192.168.1.52:137
210.x.x.23:1035	192.168.6.191:137
210.x.x.23:1035	192.168.1.241:137
210.x.x.23:1035	192.168.2.91:137
210.x.x.23:1035	192.168.1.5:137

scanning on TCP port 139.

TCP ports 80 and 443 are used by the HTTP and the HTTPS (HTTP over Secure Socket Layer), respectively. None of the ChinaSat users deployed web servers on either ports. In the collected traces, the users either ignore the requests on these two ports or reply back with TCP RSTs. On Dec. 14, 2002, there were over 15 external addresses scanning on port 80 and one external address scanning on port 443.

Ports scans on TCP port 1433 were directed at *MS SQL* servers. Even though our traces were recorded months before the well-known SQL Slammer worm was released, another *MS SQL* vulnerability was discovered in Oct. 2002, the "HELLO authentication buffer overflow" [75]. We suspect that the scans detected from the Dec. 14 traces were related to this vulnerability.

There were also port scans on TCP port 27374. The only known application that utilizes this port is the *SubSeven* trojan. Hence, scans directed toward TCP 27374 are intended to discover compromised systems. Interestingly, the SYN packets originating from the three external addresses also contain invalid TCP options. These options are listed in the following order: MSS, timestamp, end of options (EOL), other options. The EOL TCP option signals the end of the TCP options list but is rarely used in TCP/IP implementations. Instead of EOLs, NOPs are usually employed. Furthermore, no other option should be present after the EOL option. Similar to packets with invalid TCP flags, these scans may be used to exploit vulnerabilities in TCP/IP implementations.

5.4.4 Traffic volume anomalies

Wavelet decomposition of data traffic has been used to identify anomalies in traffic volume [16], [19]. We record packet count and bytes from the collected *tcpdump* traces and also employ wavelet decomposition to analyze 226,390 seconds of binned data. The data is decomposed into 12 levels by employing the *Debauchies 9* mother wavelet in MATLAB. Each wavelet coefficient at the coarsest level approximately represents 6 minutes of traffic $(226,390/569 \times 60)$ with a shape similar to that of the billing data. The approximation of the downloaded packets at the coarsest level (a_{12}) is shown in Figure 5.8. Detail coefficients from level 12 (d_{12}, coarsest) to level 1 (d_1, finest) are shown in Figures 5.9 – 5.20, respectively.

Multiple spikes observed in Figure 5.9 correspond to large variations in the hourly traffic volume. The spikes are detected by using a moving window of length 20 and calculating the standard deviation for each window. Anomalies in traffic volume are indicated by wavelet coefficients that are above or below 3 σ value (three times the standard deviation). The 3 σ lines are shown in Figure 5.9 – 5.15 for wavelet coefficients $d_1 2$ to d_6. An example of an anomaly is the one detected on Jan. 3, 2003. It was due to a network outage and recovery that was also detected in the billing records.

Finer detail levels can be used to detect anomalies at various time scales. The detail coefficients d_6 represents the time scale of one minute shown in Figure 5.15. The anomaly detected on Dec. 19, 2002 cannot be observed from the coarser level coefficients (such as d_{12}). This anomaly was first detected at level 8. The anomaly was caused by port scans that lasted approximately five minutes.

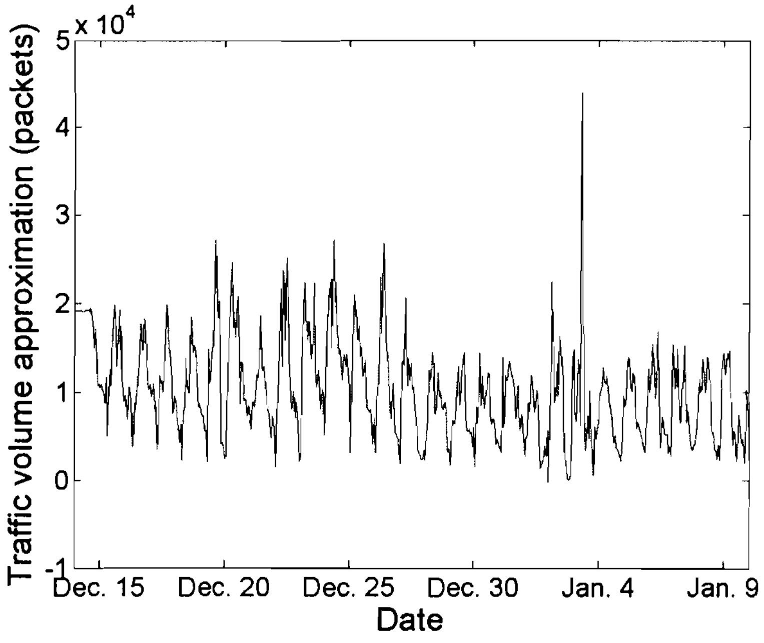

Figure 5.8. Wavelet approximation of the *tcpdump* trace (downloaded packets) at the coarsest time scale (a_{12}).

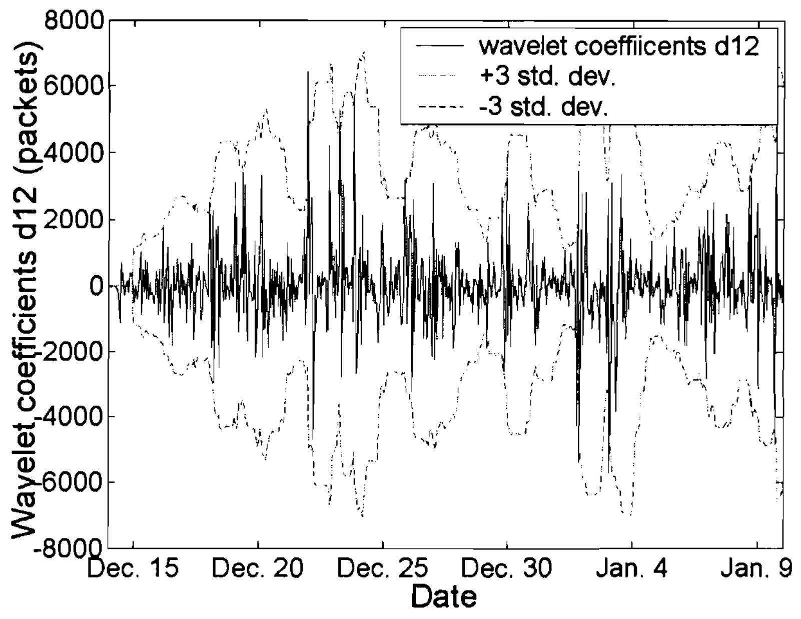

Figure 5.9. Detail wavelet coefficients d_{12} of the *tcpdump* trace (downloaded packets) at the coarsest level. Each coefficient represents 6 minutes of traffic.

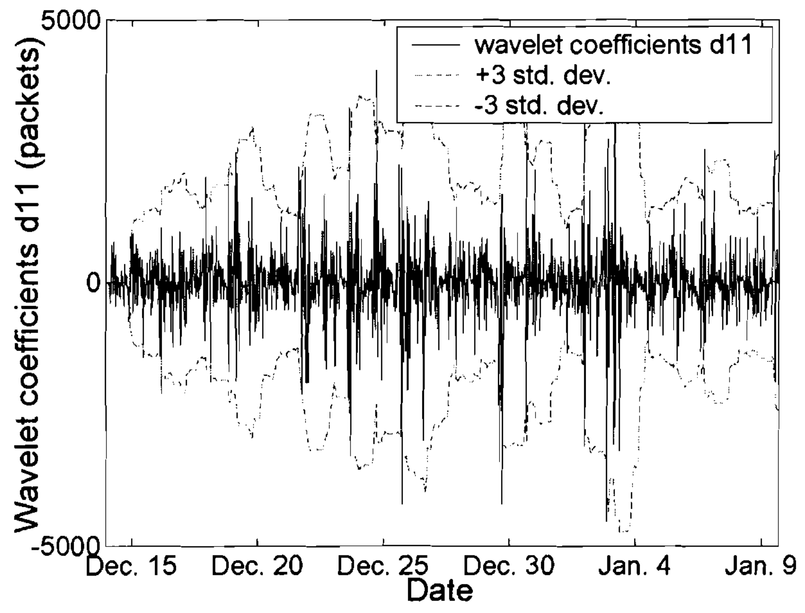

Figure 5.10. Detail wavelet coefficients d_{11} of the *tcpdump* trace (downloaded packets) at level 11.

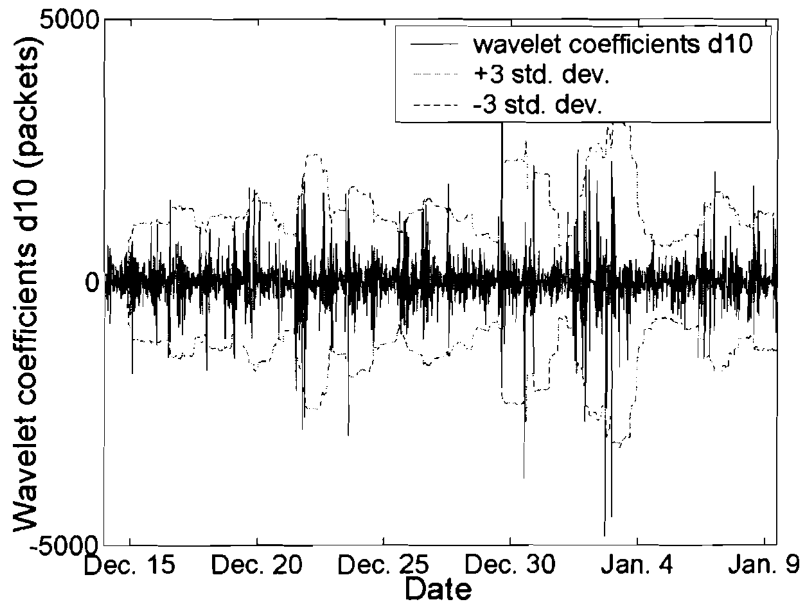

Figure 5.11. Detail wavelet coefficients d_{10} of the *tcpdump* trace (downloaded packets) at level 10.

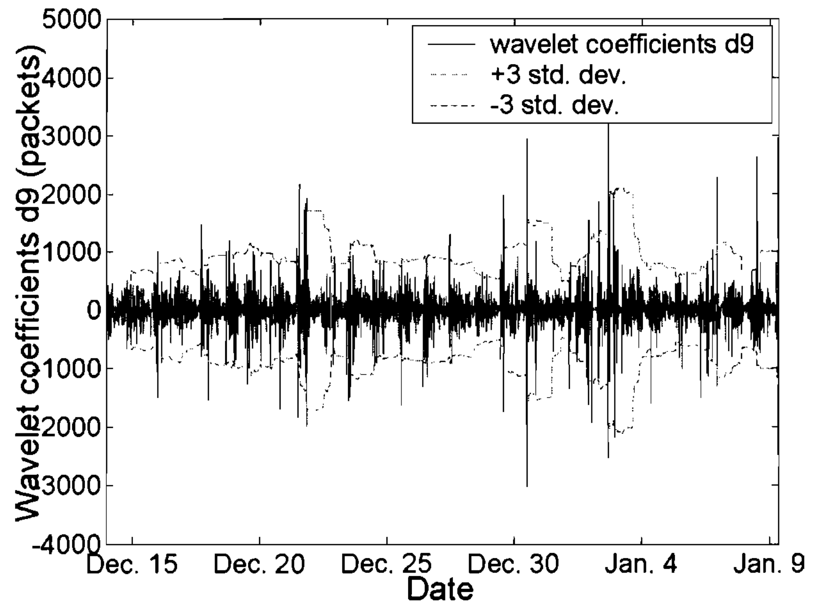

Figure 5.12. Detail wavelet coefficients d_9 of the *tcpdump* trace (downloaded packets) at level 9.

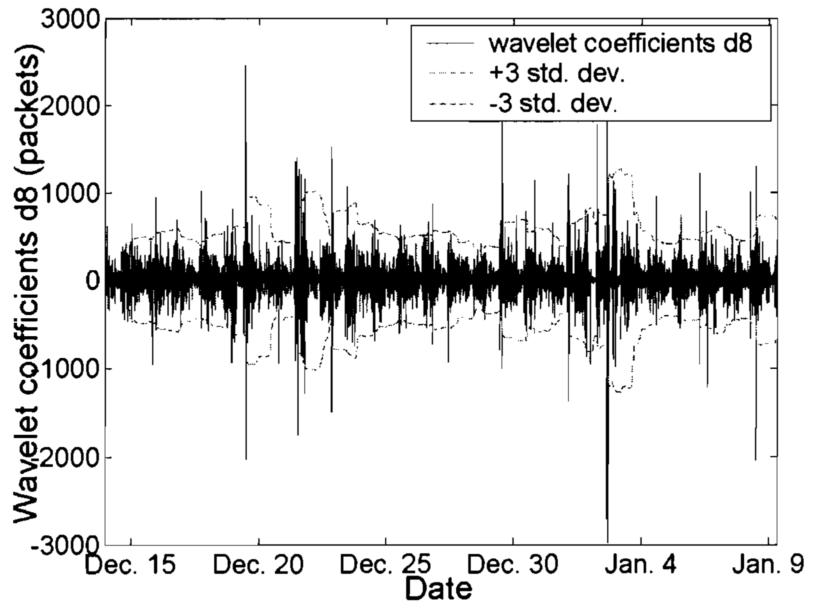

Figure 5.13. Detail wavelet coefficients d_8 of the *tcpdump* trace (downloaded packets) at level 8.

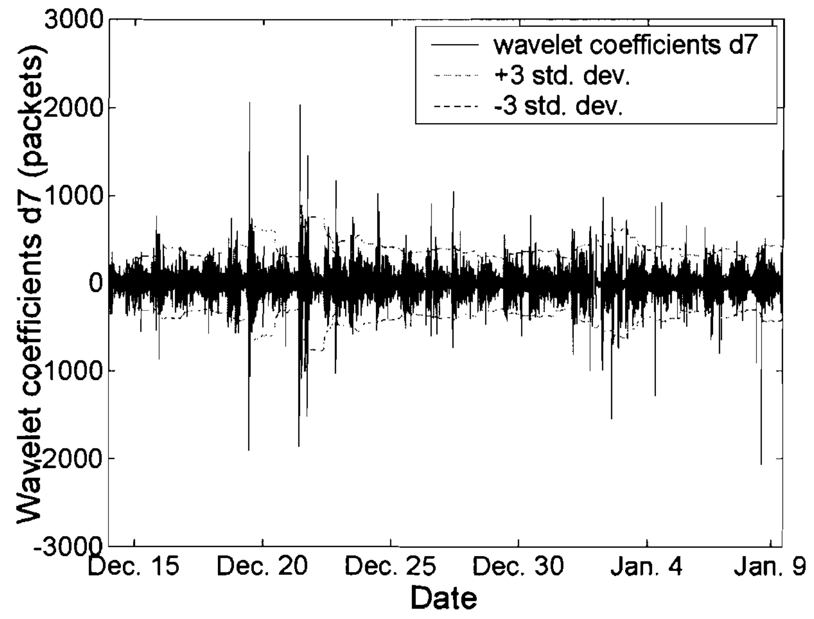

Figure 5.14. Detail wavelet coefficients d_7 of the *tcpdump* trace (downloaded packets) at level 7.

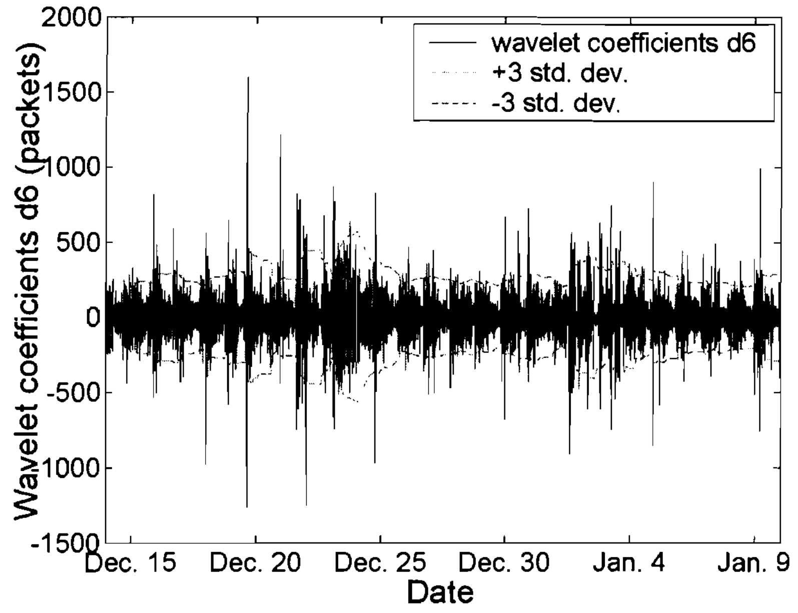

Figure 5.15. Detail wavelet coefficients d_6 of the *tcpdump* trace (downloaded packets) at level 6 (time scale equals to 26 seconds).

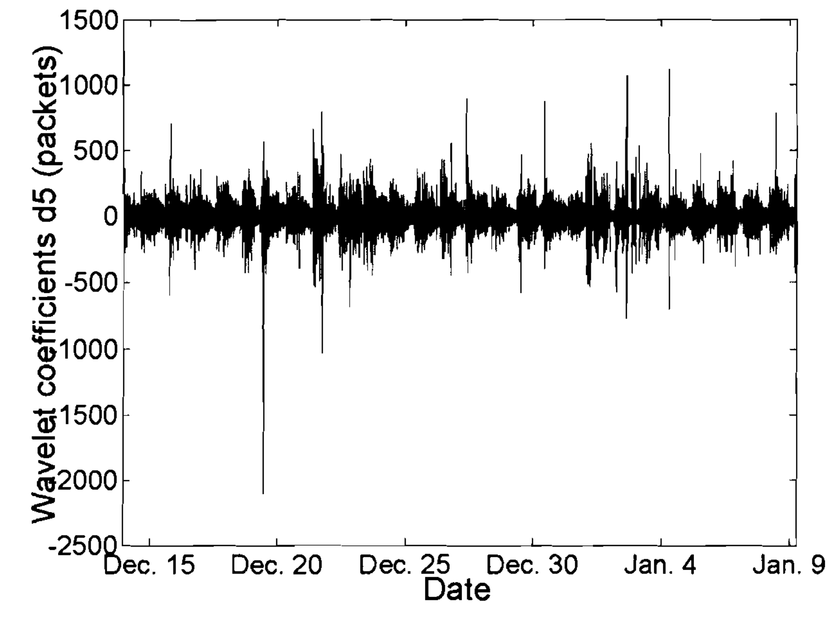

Figure 5.16. Detail wavelet coefficients d_5 of the *tcpdump* trace (downloaded packets) at level 5.

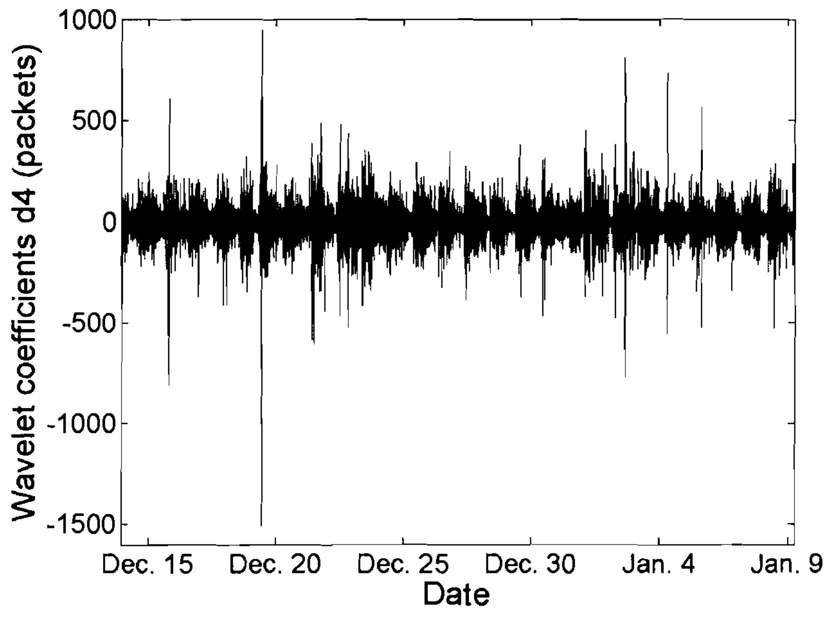

Figure 5.17. Detail wavelet coefficients d_4 of the *tcpdump* trace (downloaded packets) at level 4.

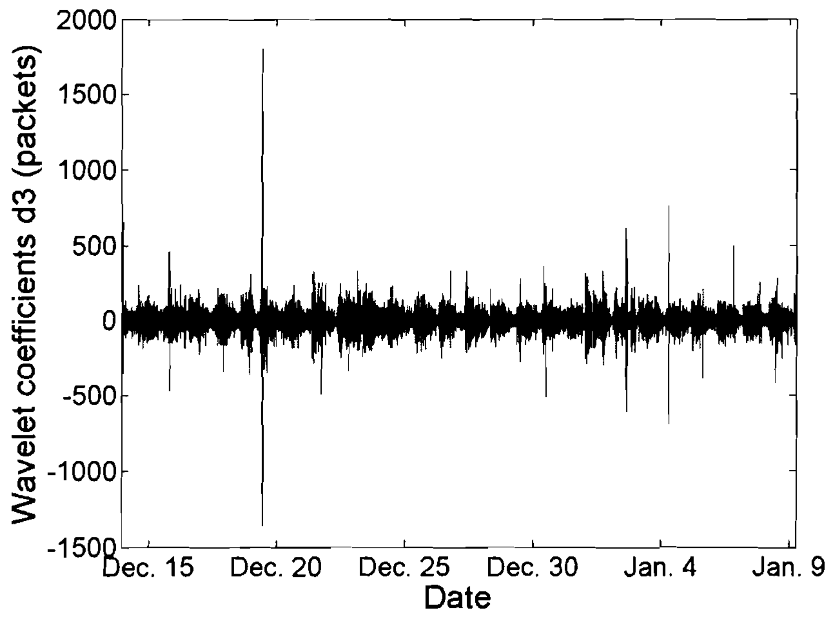

Figure 5.18. Detail wavelet coefficients d_3 of the *tcpdump* trace (downloaded packets) at level 3.

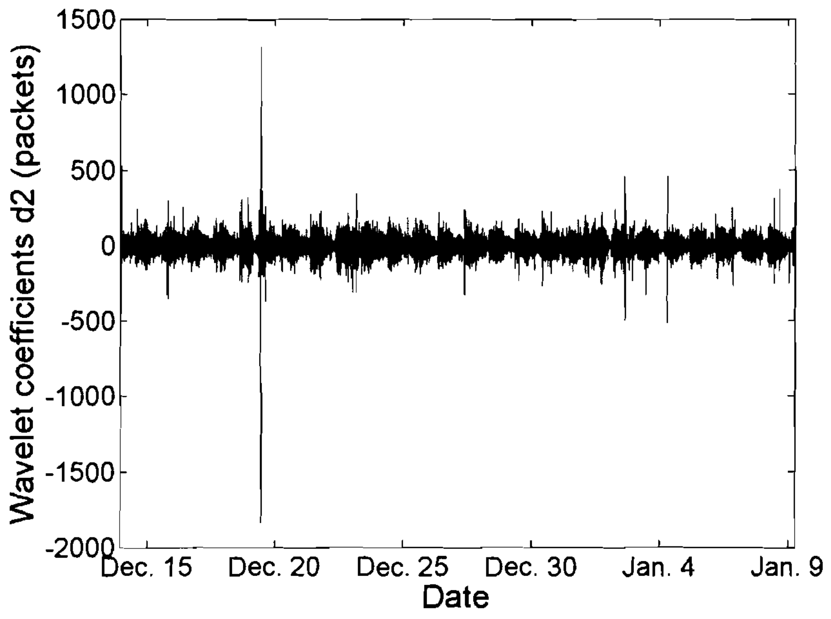

Figure 5.19. Detail wavelet coefficients d_2 of the *tcpdump* trace (downloaded packets) at level 2.

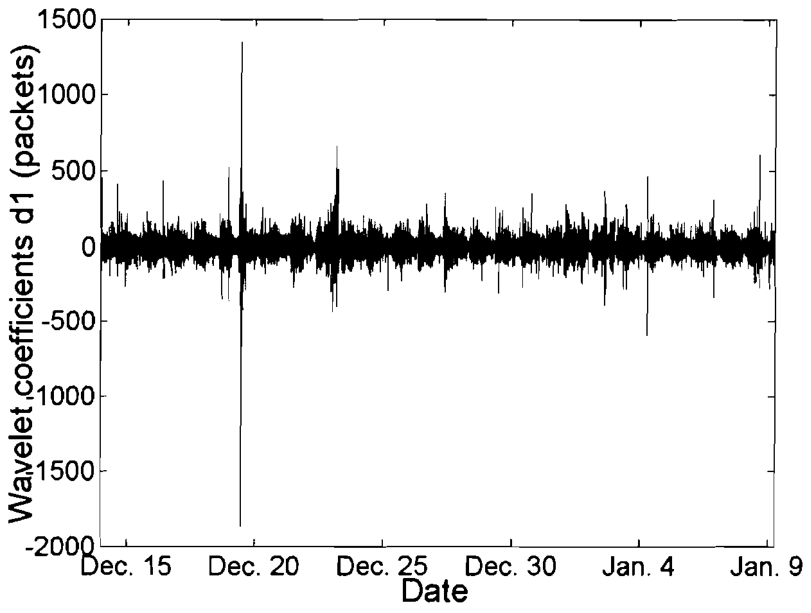

Figure 5.20. Detail wavelet coefficients d_1 of the *tcpdump* trace (downloaded packets) at the finest level (level 1).

Chapter 6

Conclusions and future work

In this thesis, we described traffic collection in a commercial hybrid satellite-terrestrial network and analyzed the billing records and collected traffic traces. The billing records indicate that the downloaded and uploaded traffic patterns were highly regular, exhibiting both daily and weekly cycles. A daily minimum occurs at 7 AM while three daily maxima occur at 11 AM, 3 PM, and 7 PM. A minority of users contributed the majority of traffic. k-means and hierarchical clustering were employed to classify the users. k-means clustering indicated that the natural number of clusters is 2 for both downloaded and uploaded packets and 3 for both downloaded and uploaded bytes, respectively. We also employed hierarchical clustering to group users by their traffic patterns. The use of inconsistency coefficients resulted in 64 clusters. We further refined our results by clustering with the three most common traffic patterns: *inactive*, *active*, and *semi-active*. Most users were found to be *inactive*.

Analysis of tcpdump traces showed that the trace is dominated by TCP traffic, with HTTP/WWW packets contributing to the majority of captured data. By examining the TCP SYN packets, we determined that SACK and increasing initial windows size were the TCP options most widely used to improve performance in the ChinaSat network. Based on this result, we propose that the hosts in the ChinaSat DirecPC network may be further optimized by ensuring the SACK option is enabled and by enabling the sliding window scale option. We also detected data traffic anomalies using open source tools and wavelet decomposition. The anomalies included invalid TCP flag combinations, large number of TCP resets, port scans, and abnormal changes in traffic volume. We provided plausible explanations for the origin of these anomalies.

Further analysis of the ChinaSat data work may focus on using patterns recognition techniques to classify users without the quantization of the traffic data. *tcpdump* traces could also be further examined in detail to investigate the effects of illegitimate traffic on the performance of the ChinaSat network.

Analysis techniques described in this thesis may be applied to data captured from other deployed networks. Such results may be used to compare the difference in performance and user behavior between ChinaSat and other networks. Lastly, if additional billing records and traffic traces could be obtained, it would be worthwhile to compare the analysis of traffic data from the newly deployed DirecWay network [24] with the results presented in this thesis.

Reference List

[1] The Internet traffic archive. [Online]. Available: http://ita.ee.lbl.gov/.

[2] S. Jaiswal, G. Iannaccone, C. Diot, J. Kurose, and D. Towsley, "Inferring TCP connection characteristics through passive measurements," in *Proc. INFOCOM 2004*, Hong Kong, HK, Mar. 2004, pp. 1582–1592.

[3] S. McCreary and K. Claffy, "Trends in wide area IP traffic patterns," in *Proc. 13th ITC Specialist Semin. on Meas. and Modeling of IP Traffic*, Monterey, CA, Sept. 2000, pp. 1–11.

[4] K. Thompson, G. J. Miller, and R. Wilder, "Wide-area Internet traffic patterns and characteristics," *IEEE Netw.*, vol. 11, no. 6, pp. 10–23, 1997.

[5] D. Tang and M. Baker, "Analysis of a metropolitan-area wireless network," in *Proc. ACM MobiCom '99*, Seattle, WA, Sept. 1999, pp. 13–23.

[6] K. Park and W. Willinger, *The Internet As a Large-Scale Complex System.* New York, NY: Oxford University Press, 2005.

[7] D. Kotz and K. Essien, "Analysis of a campus-wide wireless network," *Wireless Networks*, vol. 11, no. 1-2, pp. 115–133, Jan. 2005.

[8] S. Sarvotham, R. Riedi, and R. Baraniuk, "Connection-level analysis and modeling of network traffic," in *Proc. ACM SIGCOMM Internet Meas. Workshop 2001*, Nov. 2001, pp. 99–103.

[9] H. Kruse, M. Allman, J. Griner, and D. Tran, "Experimentation and modelling of HTTP over satellite channels," *Int. J. of Satellite Commun.*, vol. 19, no. 1, pp. 51–68, Jan.Feb. 2001.

[10] B. Vujičić, H. Chen, and Lj. Trajković, "Prediction of traffic in a public safety network," in *Proc. IEEE Int. Symp. Circuits and Systems 2006*, Kos, Greece, May 2006, pp. 2637–2640.

[11] V. G. Bharadwaj, J. S. Baras, and N. P. Butts, "An architecture for Internet service via broadband satellite networks," *Int. J. of Satellite Commun.*, vol. 19, no. 1, pp. 29–50, Jan./Feb. 2001.

[12] T. R. Henderson and R. H. Katz, "Transport protocols for Internet-compatible satellite networks," *IEEE J. Sel. Areas Commun.*, vol. 17, no. 2, pp. 326–344, Feb. 1999.

[13] H. Balakrishnan, V. Padmanabhan, and R. H. Katz, "The effects of asymmetry on TCP performance," in *Proc. ACM/IEEE MobiCom '97*, Budapest, Hungary, Sept. 1997, pp. 77–89.

[14] Q. Shao and Lj. Trajković, "Measurement and analysis of traffic in a hybrid satellite-terrestrial network," in *Proc. SPECTS 2004*, San Jose, CA, July 2004, pp. 329–336.

[15] P. Barford and D. Plonka, "Characteristics of network traffic flow anomalies," in *Proc. ACM SIGCOMM Internet Meas. Workshop 2001*, Nov. 2001, pp. 69–73.

[16] P. Barford, J. Kline, D. Plonka, and A. Ron, "A signal analysis of network traffic anomalies," in *Proc. ACM SIGCOMM Internet Meas. Workshop 2002*, Marseille, France, Nov. 2002, pp. 71–82.

[17] Y. Zhang, Z. Ge, A. Greenberg, and M. Roughan, "Network anomography," in *Proc. ACM SIGCOMM Internet Meas. Conf. 2005*, Berkeley, CA, Oct. 2005, pp. 317–330.

[18] A. Soule, K. Salamatian, and N. Taft, "Combining filtering and statistical methods for anomaly detection," in *Proc. ACM SIGCOMM Internet Meas. Conf. 2005*, Berkeley, CA, Oct. 2005, pp. 331–344.

[19] P. Huang, A. Feldmann, and W. Willinger, "A non-instrusive, wavelet-based approach to detecting network performance problems," in *Proc. ACM SIGCOMM Internet Meas. Workshop 2001*, San Francisco, CA, Nov. 2001, pp. 213–227.

[20] A. Lakhina, M. Crovella, and C. Diot, "Diagnosing network-wide traffic anomalies," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 34, no. 4, pp. 219–230, Oct. 2004.

[21] tcpdump. [Online]. Available: http://www.tcpdump.org/.

[22] S. Lau and Lj. Trajkovic, "Analysis of traffic data from a hybrid satellite-terrestrial network," to be presented at *The Fourth International Conference*

on Heterogeneous Networking for Quality, Reliability, Security, and Robustness (QShine 2007), Vancouver, Canada, Aug. 2007.

[23] J. Postel, Ed., "Transmission Control Protocol," RFC 793, Sept. 1981.

[24] "The DirecWay system," Hughes Network Systems. [Online]. Available: http://www.direcway.com.

[25] J. Postel, Ed., "Internet Protocol," RFC 791, Sept. 1981.

[26] B. R. Elbert, *Introduction to Satellite Communication*, 2nd ed. Norwood, MA: Artech House, 1999.

[27] M. Allman, D. Glover, and L. Sanchez, "Enhancing TCP over satellite channels using standard mechanisms," RFC 2488, Jan. 1999.

[28] Y. Shang and M. Hadjitheodosiou, "TCP splitting protocol for broadband and aeronautical satellite network," in *Proc. 23rd IEEE Digital Avionics Syst. Conf.*, Salt Lake City, UT, Oct. 2004, vol. 2, pp. 11.C.3-1–11.C.3-9.

[29] V. Jacobson, R. Braden, and D. Borman, "TCP extensions for high performance," RFC 1323, May 1992.

[30] M. Allman, S. Dawkins, D. Glover, J. Griner, D. Tran, T. Henderson, J. Heidemann, J. Touch, H. Kruse, S. Ostermann, K. Scott, and J. Semke, "Ongoing TCP research related to satellites," RFC 2760, Feb. 2000.

[31] J. Border, M. Kojo, J. Griner, G. Montenegro, and Z. Shelby, "Performance enhancing proxies intended to mitigate link-related degradations," RFC 3135, June 2001.

[32] S. Oueslati-Boulahia, A. Serhrouchni, S. Tohmé, S. Baier, and M. Berrada, "TCP over satellite links: problems and solutions," *Telecommun. Syst.*, vol. 13, no. 2–4, pp. 199–212, July 2000.

[33] M. Omueti and Lj. Trajkovic, "TCP with adaptive delay and loss response for heterogeneous networks," to be presented at *Wireless Internet Conf. (WICON) 2007*, Vancouver, Canada, Aug. 2007.

[34] M. Allman, S. Floyd, and C. Partridge, "Increasing TCP's initial window," RFC 2414, Sept. 1998.

[35] M. Mathis, J. Mahdavi, S. Floyd, and A. Romanow, "TCP selective acknowledgment options," RFC 2018, Oct. 1996.

[36] J. Mogul and S. Deering, "Path MTU discovery," RFC 1191, Nov. 1990.

[37] J. Postel, "Internet Control Message Protocol," RFC 792, Sept. 1981.

[38] J. S. Baras, S. Corson, S. Papademetriou, I. Secka, and N. Suphasindhu, "Fast asymmetric Internet over wireless satellite-terrestrial networks," in *Proc. MILCOM '97*, Monterey, CA, Nov. 1997, pp. 372–377.

[39] J. Ishac and M. Allman, "On the performance of TCP spoofing in satellite networks," in *Proc. MILCOM 2001*, Vienna, VA, Oct. 2001, pp. 700–704.

[40] A. Lakhina, M. Crovella, and C. Diot, "Characterization of network-wide anomalies in traffic flows," in *Proc. ACM SIGCOMM Internet Meas. Conf. 2004*, Taormina, Italy, Oct. 2004, pp. 201–206.

[41] J. Han and M. Kamber, *Data Mining: concept and techniques.* San Diego, CA: Academic Press, 2001.

[42] W. Wu, H. Xiong, and S. Shekhar, *Clustering and Information Retrieval.* Norwell, MA: Kluwer Academic Publishers, 2004.

[43] Z. Chen, *Data Mining and Uncertainty Reasoning: and integrated approach.* New York, NY: John Wiley & Sons, 2001.

[44] T. Kanungo, D. M. Mount, N. Netanyahu, C. Piatko, R. Silverman, and A. Y. Wu, "An efficient k-means clustering algorithm: analysis and implementation," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 24, no. 7, pp. 881–892, July. 2002.

[45] P.-N. Tan, M. Steinbach, and V. Kumar, *Introduction to Data Mining.* Reading, MA: Addison-Wesley, 2006, pp. 487–568.

[46] L. Kaufman and P. J. Rousseeuw, *Finding Groups in Data: an introduction to cluster analysis.* New York, NY: John Wiley & Sons, 1990.

[47] A. Jain and R. Dubes, *Algorithms for Clustering Data.* Englewood Cliffs, N.J.: Prentice Hall, 1988.

[48] H. C. Romesburg, *Cluster Analysis for Researchers.* Morrisville, N.C.: Lulu press, 2004.

[49] C. K. Chui, *An Introduction to Wavelets.* San Diego, CA: Academic Press Professional, Inc., 1992.

[50] R. Carmona, W. Hwang, and B. Torrésani, *Practical Time-Frequency Analysis: continuous wavelet and Gabor transforms, with an implementation in S*, ser. Wavelet Analysis and its Applications. San Diego, CA: Academic Press, 1998, vol. 9.

[51] Y. Y. Tang, L. H. Yang, J. Liu, and H. Ma, Eds., *Wavelet Theory and Its Application to Pattern Recognition.* Singapore: World Scientific Publishing Co. Pte. Ltd., 2000.

[52] *DirecPC Product Technical Specification, Release 2.1*, Hughes Network Systems, 1999.

[53] MATLAB. [Online]. Available: http://www.mathworks.com/products/matlab/.

[54] V. Paxson, "Empirically derived analytic models of wide-area TCP connections," *IEEE/ACM Trans. Netw.*, vol. 2, no. 4, pp. 316–336, Aug. 1994.

[55] W.-K. Ching and M. K.-P. Ng, Eds., *Advances in Data Mining and Modeling.* Singapore: World Scientific Publishing Co. Pte. Ltd., 2003.

[56] M. Last, A. Kandel, and H. Bunke, Eds., *Data Mining in Time Series Databases.* Singapore: World Scientific Publishing Co. Pte. Ltd., 2004.

[57] D. E. Comer, *Internetworking with TCP/IP, Vol 1: Principles, Protocols, and Architecture*, 4th ed. Upper Saddle River, NJ: Prentice-Hall, 2000.

[58] W. R. Stevens, *TCP/IP Illustrated (vol. 1): The Protocols.* Reading, MA: Addison-Wesley, 1994.

[59] Y. Rekhter, B. Moskowitz, D. Karrenberg, G. J. de Groot, and E. Lear, "Address allocation for private Internets," RFC 1918, Feb. 1996.

[60] K. Egevang, "The IP network address translator (NAT)," RFC 1631, May 1994.

[61] R. Droms, "Dynamic Host Configuration Protocol," RFC 2131, Mar. 1997.

[62] D. C. Plummer, "An Ethernet address reolution protocol," RFC 826, Nov. 1982.

[63] R. Finlayson, T. Mann, J. Mogul, and M. Theimer, "A reverse address resolution protocol," RFC 903, June 1984.

[64] G. Malkin, "RIP version 2," RFC 2453, Nov. 1998.

[65] Microsoft Windows 2000 TCP/IP implementation details. [Online]. Available: http://www.microsoft.com/technet/itsolutions/network/deploy/depovg /tcpip2k.mspx.

[66] R. Beverly, "A robust classifier for passive TCP/IP fingerprinting," in *Proc. Passive and Active Meas. Workshop 2004*, Antibes Juan-les-Pins, France, Apr. 2004, pp. 158–167.

[67] C. Smith and P. Grundl, "Know your enemy: passive fingerprinting," The Honeynet Project, Mar. 2002. [Online]. Available: http://www.honeynet.org/papers/finger/.

[68] Passive OS fingerprinting tool ver. 2 (p0f v2). [Online]. Available: http://lcamtuf.coredump.cx/p0f.shtml.

[69] B. Petersen, "Intrusion detection FAQ: what is p0f and what does it do?" The SysAdmin, Audit, Network, Security (SANS) Institute. [Online]. Available: http://www.sans.org/resources/idfaq/p0f.php.

[70] T. Miller, "Passive OS fingerprinting: details and techniques," The SysAdmin, Audit, Network, Security (SANS) Institute. [Online]. Available: http://www.sans.org/reading_room/special.php.

[71] Wireshark (formerly Ethereal). [Online]. Available: http://www.wireshark.org/.

[72] tcptrace. [Online]. Available: http://jarok.cs.ohiou.edu/software/tcptrace/.

[73] J. Postel, "TCP and IP bake off," RFC 1025, Sept. 1987.

[74] M. Arlitt and C. Williamson, "An analysis of TCP reset behaviour on the Internet," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 35, no. 1, pp. 37–44, Jan. 2005.

[75] Microsoft Security Bulletin MS02-056, October 2002. [Online]. Available: http://www.microsoft.com/technet/security/bulletin/MS02-056.mspx.

Appendix A

Code listing

A.1 Pre-processing code

A.1.1 *normalize.m*

```
%Normalizetime function
%Input: a processed version of the ChinaSat billing data
%(with invalid entries removed)
%Output: returns the earliest time in the data set (baseline)
%and also returns norm_time_data, which is the delimiteddata
%matrix augmented with 7 columns attached to the end.
%
%   baseline is a 1x7 matrix consisting of the following:
%      (1,1): the earliest START_TIME timestamp recording
%      in the billing data format (ex. 20021031230007)
%      (1,2): the 4 digit year value from the START_TIME
%      timestamp recorded in (1,1) (ex. 2002)
%      (1,3): the 2 digit month value from the START_TIME
%      timestamp recorded in (1,1) (ex. 10)
%      (1,4): the 2 digit day value from the START_TIME
%      timestamp recorded in (1,1) (ex. 31)
%      (1,5): the 2 digit hour (24 hr based) value from
%      the START_TIME timestamp recorded in (1,1) (ex. 23)
%      (1,6): the value, in hours, from January first of
%      the year recorded in (1,1) (ex. 7319)
%      (1,7): the value, in days, from January first of
%      the year recorded in (1,1) (ex. 304)
%
%   The first 5 columns augmented to norm_time_data has
%   the same description as the columns (1,2) to (1,7),
%   with the START_TIME timestamp find in the 4th
%   column of the norm_time_data.  The 6th added column
%   contains the value of the the 5th and 6th added column
%   in norm_time_data subtracted by the (1,6) and (1,7)
%   value in baseline, respectivcely.

%   Thus, the value stored in this 6th column is the
%   difference in hours from the first recorded START_TIME
%   timestamp.  For example, an timestamp with the date of
%   20021101000055 (Nov 1st, 2002, 0000 hours) will have a
%   value of 7320 in the 5th added column.  The value in the
%   6th added column will be 1, since it is 1 hour away from
%   the starting time of 20021031230007.
function [norm_time_data,baseline] = normalizetime(delimiteddata)

%Create a matrix called norm_time_data that is 6 columns wider than
%delimiteddata.  Copy delimiteddata to the first 11 columns of
%norm_time_data.
norm_time_data=zeros(size(delimiteddata,1),size(delimiteddata,2)+8);
norm_time_data(:,1:11)=delimiteddata(:,1:11);

%Creates the baseline variable and find the earliest time stamp
```

```
% recorded in the norm_time_data's START_TIME column.
% baseline (1,2) to (1,5) are the year, month, day, and hour
%value in the START_TIME, respectively.
baseline = zeros(1,7);
baseline(1,1) = min(norm_time_data(:,4));
baseline(1,2) = floor(baseline(1,1)/1e10);
baseline(1,3) = floor(rem(baseline(1,1)/1e8,100));
baseline(1,4) = floor(rem(baseline(1,1)/1e6,100));
baseline(1,5) = floor(rem(baseline(1,1)/1e4,100));

%The switch statement returns temp, the number of days in the year
%preceding the start of the month value recorded in baseline(1,3).
%The code commented out is not used since the obtained data set do
%not contain any leap years.

    switch baseline(1,3)
        case (1)
            temp = 0;
        case (2)
            temp = 31;
        case (3)
            temp = 59;
        case (4)
            temp = 90;
        case (5)
            temp = 120;
        case (6)
            temp = 151;
        case (7)
            temp = 181;
        case (8)
```

```
                temp = 212;
            case (9)
                temp = 243;
            case (10)
                temp = 273;
            case (11)
                temp = 304;
            case (12)
                temp = 334;
        end
%end

%temp2 returns the year value difference from 2002 (expressed
%in days)
temp2 = (baseline(1,2) - 2002)*365;

%baseline(1,6) returns the number of hours since Jan 1st, 2002
%comapred to the earliest recorded START_TIME timestamp.
baseline(1,6) = baseline(1,5) + (temp2 + temp + baseline(1,4)) * 24;
baseline(1,7) = baseline(1,4)+ temp + temp2;

clear temp temp2;

%norm_time_data (:,12) to (:,15) are the year, month, day, and hour
%value in the START_TIME column (:,4), respectively.
norm_time_data(:,12)=floor(norm_time_data(:,4)/1e10);
norm_time_data(:,13)=floor(rem(norm_time_data(:,4)/1e8,100));
norm_time_data(:,14)=floor(rem(norm_time_data(:,4)/1e6,100));
norm_time_data(:,15)=floor(rem(norm_time_data(:,4)/1e4,100));

%The switch statement returns temp, the number of days in the year
```

```
%preceding the start of the month value recorded in the 4th column
%(START_TIME timestamp).
for i = 1:size(norm_time_data,1)
    switch norm_time_data(i,13)
        case (1)
            temp = 0;
        case (2)
            temp = 31;
        case (3)
            temp = 59;
        case (4)
            temp = 90;
        case (5)
            temp = 120;
        case (6)
            temp = 151;
        case (7)
            temp = 181;
        case (8)
            temp = 212;
        case (9)
            temp = 243;
        case (10)
            temp = 273;
        case (11)
            temp = 304;
        case (12)
            temp = 334;
    end

%temp2 returns the year value difference from 2002
```

```
%(expressed in days)
temp2 = (norm_time_data(i,12) - 2002)*365;

%norm_time_data(i,16) returns the number of hours since
%Jan 1st, 2002 compared to the START_TIME timestamp
%in the 4th column.
norm_time_data(i,16) = norm_time_data(i,15)
  + (temp2 + temp + norm_time_data(i,14)) * 24;

%norm_time_data(i,17) returns the difference in hours
%compared to the value recorded in baseline(1,6)
norm_time_data(i,17) = norm_time_data(i,16) - baseline(1,6);
norm_time_data(i,18) = norm_time_data(i,14) + temp + temp2;
norm_time_data(i,19) = norm_time_data(i,18) - baseline(1,7);

end
```

A.1.2 *mergebilling.m*

```
%mergebilling function
%Input: the data set (norm_time_data) with normalized time
%(from normalizetime function)
%Output: returns the the merged billing data (merged_data)
%and the aggregated data (agg_data)
%
%   merged_data is a truncated version of norm_time_data.
%   As there are two billing data collection points,
%   some users' activities may be recorded on two separate
%   files for the same hour.  This function combines the
%   entries that have the same SiteID and START_TIME.
%
```

```
%   merged_data has 11 column and the following statistics,
%   in order: SiteID, normalized hour value, Cmin, CTxByte,
%   CRxByte, CTxPkt, CRxPkt, year, month, day and hour.
%
%   agg_data is the SiteID-aggregated data.  All SiteID
%   statistics from the same hour are combined into one
%   entry.  agg_data has 6 columns in the following order:
%   normalized hour, Cmin, CTxByte, CRxByte, CTxPkt, CRxPkt.
%
%   CTxByte (Tx byte count), CRxByte (Rx byte count),
%   CTxPkt (Tx packet count) and CRxPkt (Rx packet count)
%   are combined by additions. Cmin is combined by taking the
%   minimum of (Cmin sum, 60).  Note that the ChinaSat dataset
%   contains entries where the Cmin > 60.  Those entries
%   are truncated to be 60.
function [merged_data,agg_by_hour,agg_by_date,temp] =
mergebilling(norm_time_data)

%copy and arrange norm_time_data to the desired arrangement
temp5 = zeros(size(norm_time_data,1),12);
temp5(:,1) = norm_time_data(:,3);
temp5(:,2) = norm_time_data(:,17);
temp5(:,3) = norm_time_data(:,19);
temp5(:,4) = norm_time_data(:,6);
temp5(:,5) = norm_time_data(:,8);
temp5(:,6) = norm_time_data(:,9);
temp5(:,7) = norm_time_data(:,10);
temp5(:,8) = norm_time_data(:,11);
temp5(:,9) = norm_time_data(:,12);
temp5(:,10) = norm_time_data(:,13);
temp5(:,11) = norm_time_data(:,14);
```

```
temp5(:,12) = norm_time_data(:,15);

%Sort temp5, the rearranged norm_time_data matrix by siteID
temp5 = sortrows(temp5,1);
temp = zeros(size(temp5,1),size(temp5,2));

%Combine the rows that have same SiteID and normalized hour
%value
temp(1,:)=temp5(1,:);
j = 2;
for i = 2:size(temp5,1)
    if (temp(j-1,1) == temp5(i,1)) && (temp(j-1,2) == temp5(i,2))
        temp(j-1,4) = min(60,temp(j-1,4)+min(60,temp5(i,4)));
        temp(j-1,5) = temp(j-1,5)+temp5(i,5);
        temp(j-1,6) = temp(j-1,6)+temp5(i,6);
        temp(j-1,7) = temp(j-1,7)+temp5(i,7);
        temp(j-1,8) = temp(j-1,8)+temp5(i,8);
    else
        temp(j,:) = temp5(i,:);
        temp(j,4) = min(60,temp(j,4));
        j=j+1;
    end
end

%Remove the emptry rows
filled_rows = max(find(temp(:,1)));
merged_data = zeros(filled_rows,size(temp5,2));
for i = 1:filled_rows
    merged_data(i,:)=temp(i,:);
end
clear temp temp5;
```

```
%Sort merged_data by normalized time value and place into temp.
temp = sortrows(merged_data,2);
temp2 = zeros(size(merged_data,1),size(merged_data,2)-1);

%Regardless of SiteID, combine all entries that have the same
%normalized hour value.
for k =2:size(merged_data,2)
    temp2(1,k-1)=temp(1,k);
end
j=2;
for i = 2:size(temp,1)
    if (temp2(j-1,1)==temp(i,2))
        temp2(j-1,3)=max(temp2(j-1,3),temp(i,4));
        temp2(j-1,4)=temp2(j-1,4)+temp(i,5);
        temp2(j-1,5)=temp2(j-1,5)+temp(i,6);
        temp2(j-1,6)=temp2(j-1,6)+temp(i,7);
        temp2(j-1,7)=temp2(j-1,7)+temp(i,8);
    else
        for k = 2:size(temp,2)
            temp2(j,k-1)=temp(i,k);
        end
        j=j+1;
    end
end
filled_rows = max(find(temp2(:,1)));
agg_by_hour = zeros(filled_rows,size(temp2,2));
for i = 1:filled_rows
    agg_by_hour(i,:)=temp2(i,:);
end
```

```
clear temp1, temp2;

temp = sortrows(agg_by_hour,2);
temp2 = zeros(size(agg_by_hour,1),size(agg_by_hour,2)-1);

%Regardless of SiteID, combine all entries that have the same
%normalized hour value.
for k =2:size(agg_by_hour,2)
    temp2(1,k-1)=temp(1,k);
end
j=2;
for i = 2:size(temp,1)
    if (temp2(j-1,1)==temp(i,2))
        temp2(j-1,2)=max(temp2(j-1,2),temp(i,3));
        temp2(j-1,3)=temp2(j-1,3)+temp(i,4);
        temp2(j-1,4)=temp2(j-1,4)+temp(i,5);
        temp2(j-1,5)=temp2(j-1,5)+temp(i,6);
        temp2(j-1,6)=temp2(j-1,6)+temp(i,7);
    else
        for k = 2:size(temp,2)
            temp2(j,k-1)=temp(i,k);
        end
        j=j+1;
    end
end
filled_rows = max(find(temp2(:,1)));
agg_by_date = zeros(filled_rows,size(temp2,2));
for i = 1:filled_rows
    agg_by_date(i,:)=temp2(i,:);
end
```

A.2 *pcapread* code

A.2.1 *pcapread.h*

```
#include <stdio.h>
#include <string>
#include <time.h>
#include <getopt.h>

#define ETH_ADDR_LEN 6 /* Ethernet address length */
#define IP_ADDR_LEN 4 /* IP address length */
#define IP_ADDR_SKIP 12
/* IP addresses are from 12 byte of ip header */
#define HEADER_LEN (6+6+2+IP_ADDR_SKIP+4+4)
/* 34 byte of header info of record */

#define IP_TYPE (0x0800)
/* 0x0800 is hte value of type field for IP packets */

#define TCP_OPT0_LEN 1
#define TCP_OPT1_LEN 1
#define TCP_OPT2_LEN 4
#define TCP_OPT3_LEN 3
#define TCP_OPT4_LEN 2
#define TCP_OPT8_LEN 10

using namespace std;
static struct option main_longopt[] = {
        { "ethernet", no_argument, 0, 'e' },
        { "tcpopt", no_argument, 0, 'o' },
        { "readfile", required_argument, 0, 'r' },
        {0, 0, 0, 0}
};

typedef unsigned char u_char;
typedef unsigned short u_short;
typedef unsigned long u_long;

typedef struct timeval TimeStamp;
long snap_len;

typedef struct
{
   TimeStamp ts; // timestamp
   int pkt_len; //Ethernet packet length
   int rec_len; //record length
   u_char src_eaddr[ETH_ADDR_LEN]; //source Ethernet address
   u_char dst_eaddr[ETH_ADDR_LEN]; //destination Ethernet address
   u_short type; // type field
   u_char ip_ver;
   u_char ip_hdr_len;
   u_char ip_tos;
   u_short ip_pkt_len;
   u_short ip_id;
   u_char ip_flags;
   u_short ip_frag_offset;
   u_char ip_ttl;
   u_char ip_proto;
   u_short ip_hdr_chksum;
   u_char src_ip_addr[IP_ADDR_LEN]; //source IP address (32 bits)
   u_char dst_ip_addr[IP_ADDR_LEN]; //destination IP address (32 bits)
   u_short tcp_src_port; // source TCP port number
   u_short tcp_dst_port;// destination TCP port number
   u_long tcp_seq_num;
   u_long tcp_ack_num;
   u_char tcp_hdr_len;
   u_short tcp_ctrl_bits;
   u_short tcp_win_size;
   u_short tcp_chksum;
   u_short tcp_urg_ptr;
   u_short udp_src_port; // source UDP port number
   u_short udp_dst_port;// destination UDP port number
   u_short udp_hdr_len;
   u_short udp_chksum;
} DumpRecord;

void ReadRecord(FILE *fp, DumpRecord *dumpRec);
void PrintRecord(DumpRecord *dumpRec);
long ReadLong(FILE *fp);
long ReadLong_r(FILE *fp);
u_short ReadShort(FILE *fp);
u_short ReadShort_r(FILE *fp);
void ReadSummary(FILE *fp);
void PrintSummary();

//ReadLong(): reads next 4 bytes from file fp, converts value
//to long and returns the value (big endian)

long ReadLong(FILE *fp)
{
   unsigned long val;

   val = (u_long)fgetc(fp);
   val = val << 8 | (u_long)fgetc(fp);
   val = val << 8 | (u_long)fgetc(fp);
   val = val << 8 | (u_long)fgetc(fp);

   return (long) val;
} //End ReadLong

//ReadLong_r(): reads next 4 bytes from file fp, converts value
//to long and returns the value (little endian)

long ReadLong_r(FILE *fp)
{
   unsigned long val;

   val = (u_long)fgetc(fp);
   val = (u_long)fgetc(fp) << 8 | val;
   val = (u_long)fgetc(fp) << 16 | val;
   val = (u_long)fgetc(fp) << 24 | val;

   return (long) val;
} //End ReadLong_r

//ReadShort(): reads next 2 bytes from file fp, converts value
//to short and returns the value (big endian)

u_short ReadShort(FILE *fp)
{
   u_short val;

   val = (u_short)fgetc(fp);
   val = val << 8 | (u_short)fgetc(fp);
   return val;
} // End ReadShort

//ReadShort_r(): reads next 2 bytes from file fp, converts value
//to short and returns the value (little endian)

u_short ReadShort_r(FILE *fp)
{
   u_short val;

   val = (u_short)fgetc(fp);
   val = (u_short)fgetc(fp) << 8 | val;

   return val;
} // End ReadShort_r

/*ReadRecord(): reads a character at time and dumps the
various fields of the next record.*/
```

A.2.2 *pcapread.c*

```
#include "pcapread.h"
/*ReadRecord(): reads a character at time and dumps the
various fields of the next record.*/

void ReadRecord(FILE *fp, DumpRecord *dumpRec)
{
   int i, ip_opt_size, tcp_opt_size, tcp_opt_size2,
    skip_length, rem_rec_len;
  u_char u_char_temp;
  u_short u_short_temp, icmp_chksum;
  u_char tcp_opt_temp, sack_size;
  u_char icmp_type, icmp_code;

  dumpRec->ts.tv_sec = ReadLong_r(fp); // read sec of timestamp
  if (feof(fp))
     return;
  dumpRec->ts.tv_usec = ReadLong_r(fp);
    // read usec of timestamp
  dumpRec->rec_len = ReadLong_r(fp);
    // recorded Ethernet packet length
  dumpRec->pkt_len = ReadLong_r(fp);
    //actual Ethernet packet length
  printf("%i %i %s",dumpRec->ts.tv_sec, dumpRec->ts.tv_usec,
    ctime(&dumpRec->ts.tv_sec));
  printf("pkt_len %i rec_len %i\n",dumpRec->pkt_len,
    dumpRec->rec_len);
  rem_rec_len = dumpRec->rec_len;
  // reading destination Ethernet address
  for (i = 0; i < ETH_ADDR_LEN; i++)
  {
     dumpRec->dst_eaddr[i] = fgetc(fp);
  }
  rem_rec_len=rem_rec_len-ETH_ADDR_LEN;
  // reading source Ethernet address
  for (i = 0; i < ETH_ADDR_LEN; i++)
  {
     dumpRec->src_eaddr[i] = fgetc(fp);
  }
```

```
rem_rec_len=rem_rec_len-ETH_ADDR_LEN;

dumpRec->type = ReadShort(fp); //read Ethernet type field
rem_rec_len=rem_rec_len-2;

// skip byte till ip address fields
u_char_temp = fgetc(fp);
rem_rec_len=rem_rec_len-1;
dumpRec->ip_ver = u_char_temp >> 4;
dumpRec->ip_hdr_len = u_char_temp & 15;
dumpRec->ip_tos = fgetc(fp);
rem_rec_len=rem_rec_len-1;
dumpRec->ip_pkt_len = ReadShort(fp);
rem_rec_len=rem_rec_len-2;
dumpRec->ip_id = ReadShort(fp);
rem_rec_len=rem_rec_len-2;
u_short_temp = ReadShort(fp);
rem_rec_len=rem_rec_len-2;
dumpRec->ip_flags = u_short_temp >> 12;
dumpRec->ip_frag_offset = u_short_temp & 4095;
printf("ip_ver %x ip_hdr_len %x ip_tos %x ip_pkt_len
  %i ip_id %i \n", dumpRec->ip_ver,dumpRec->ip_hdr_len,
  dumpRec->ip_tos, dumpRec->ip_pkt_len, dumpRec->ip_id);
printf("ip_flags %x ip_frag_offset %x \n",dumpRec->ip_flags,
  dumpRec->ip_frag_offset);
dumpRec->ip_ttl = fgetc(fp);
dumpRec->ip_proto = fgetc(fp);
dumpRec->ip_hdr_chksum = ReadShort(fp);
rem_rec_len=rem_rec_len-4;
printf("ip_ttl %u ip_proto %u ip_hdr_chksum %02x\n",
  dumpRec->ip_ttl, dumpRec->ip_proto, dumpRec->ip_hdr_chksum);
```

```
for (i = 0; i < IP_ADDR_LEN; i++)
   dumpRec->src_ip_addr[i] = fgetc(fp);
rem_rec_len=rem_rec_len-IP_ADDR_LEN;

for (i = 0; i < IP_ADDR_LEN; i++)
   dumpRec->dst_ip_addr[i] = fgetc(fp);
rem_rec_len=rem_rec_len-IP_ADDR_LEN;

printf("dumpRec->rec_len %i HEADER_LEN %i\n", dumpRec->rec_len,
  HEADER_LEN);
ip_opt_size = dumpRec->ip_hdr_len - 5;
while (ip_opt_size > 0)
{
   printf("IP packet options are:");
   for (i = 0; i < 4; i++)
   {
      printf("%02x",fgetc(fp));
      rem_rec_len=rem_rec_len-1;
   }
   printf("\n");
   ip_opt_size--;
}

if (dumpRec->ip_proto == 1)
{
   if (rem_rec_len > 4)
   {
      icmp_type=fgetc(fp);
      icmp_code=fgetc(fp);
      icmp_chksum=ReadShort(fp);
```

```
            printf("ICMP type %i code %i chksum %i\n",icmp_type,
              icmp_code, icmp_chksum);
            rem_rec_len = rem_rec_len - 4;
            printf("ICMP data ");
            for (i = 0; i < rem_rec_len; i++)
            {
                printf("%02x",fgetc(fp));
            }
            printf("\n");



           printf("rec_len %i HEADER_LEN %i ip_hdr_len %i\n",
             dumpRec->rec_len, HEADER_LEN, dumpRec->ip_hdr_len);
        }
        else
        {
           printf("truncated ICMP fields\n");
           for (i = 0; i < rem_rec_len; i++)
           {
               printf("%02x",fgetc(fp));
           }
           printf("\n");
        }
     }

     else if (dumpRec->ip_proto == 6)
     {
        dumpRec->tcp_src_port = ReadShort(fp);
        dumpRec->tcp_dst_port = ReadShort(fp);
        dumpRec->tcp_seq_num = ReadLong(fp);
        dumpRec->tcp_ack_num = ReadLong(fp);
```

```
      u_short_temp = ReadShort(fp);
      dumpRec->tcp_hdr_len = u_short_temp >> 12;
      dumpRec->tcp_ctrl_bits = u_short_temp & 63;
      dumpRec->tcp_win_size = ReadShort(fp);
      dumpRec->tcp_chksum = ReadShort(fp);
      dumpRec->tcp_urg_ptr = ReadShort(fp);

      rem_rec_len=rem_rec_len-2-2-4-4-2-2-2-2;

      tcp_opt_size = dumpRec->tcp_hdr_len - 5;
      printf("TCP: src_port %i dst_port %i seq_num %02x
        ack_num %02x hdr_len %i ctrl_bits %02x win_size
        %u chksum %02x urg_ptr %02x\n", dumpRec->tcp_src_port,
        dumpRec->tcp_dst_port, dumpRec->tcp_seq_num,
        dumpRec->tcp_ack_num, dumpRec->tcp_hdr_len,
        dumpRec->tcp_ctrl_bits, dumpRec->tcp_win_size,
        dumpRec->tcp_chksum, dumpRec->tcp_urg_ptr);
      skip_length = HEADER_LEN + 4*(dumpRec->ip_hdr_len - 5)
        + 20 + 4*(dumpRec->tcp_hdr_len - 5);

      printf("skip length %i; remaining rec_len %i\n",
        dumpRec->rec_len-skip_length, rem_rec_len);

      printf("rec_len %i HEADER_LEN %i ip_hdr_len %i skip_length
        %i\n", dumpRec->rec_len, HEADER_LEN, dumpRec->ip_hdr_len,
        skip_length);
      tcp_opt_size2 = tcp_opt_size * 4;
//      printf("tcp_opt_size2 %i \n",tcp_opt_size2);
//      if (skip_length > dumpRec->rec_len)
//      {
//         printf("tcp options truncated!\n");
//          tcp_opt_size2 = tcp_opt_size2 + dumpRec->rec_len
//            - skip_length;
//        }
          while (tcp_opt_size2 > 0 && rem_rec_len > 0)
          {
             tcp_opt_temp = fgetc(fp);
             rem_rec_len = rem_rec_len -1;
             switch (tcp_opt_temp)
             {
                case 0:
                   printf("eol\n");
                   tcp_opt_size2--;
                case 1:
                   printf("nop\n");
                   tcp_opt_size2--;
                   break;
                case 2:
                   printf("mss ");
                   if (rem_rec_len >= TCP_OPT2_LEN - 1)
                   {
                      (void)fgetc(fp);
                      ReadShort(fp);
                      printf("%i\n",ReadShort(fp));
                      tcp_opt_size2 = tcp_opt_size2 - 4;
                      rem_rec_len = rem_rec_len - 3;
                   }
                   else
                   {
                      printf("incomplete mss \n");
                      tcp_opt_size2=-1;
                   }
```

```
      break;
   case 3:
      printf("wscale ");
      if (rem_rec_len >= TCP_OPT3_LEN - 1)
      {
         (void)fgetc(fp);
         fgetc(fp);
         printf("%i\n",fgetc(fp));
         tcp_opt_size2 = tcp_opt_size2 - 3;
         rem_rec_len = rem_rec_len - 2;
      }
      else
      {
         printf("incomplete wscale \n");
         tcp_opt_size2=-1;
      }
      break;
   case 4:
      printf("SACKOK\n");
      if (rem_rec_len >= TCP_OPT4_LEN - 1)
      {
         (void)fgetc(fp);
         tcp_opt_size2 = tcp_opt_size2 - 2;
         rem_rec_len = rem_rec_len - 1;
      }
      else
      {
         printf("SACKOK incomplete \n");
         tcp_opt_size2=-1;
      }
      break;
```

```
case 5:
   printf("SACK ");
   if (rem_rec_len < 1)
   {
      tcp_opt_size2=-1;
   }
   else
   {
      sack_size = fgetc(fp);
      rem_rec_len = rem_rec_len - 1;
      if (rem_rec_len >= (sack_size-2) * 4)
      {
         printf("sacksize %i\n",sack_size);
         tcp_opt_size2 = tcp_opt_size2 - sack_size;
         sack_size = sack_size - 2;
         while (sack_size > 0)
         {
            printf("%02x",fgetc(fp));
            rem_rec_len = rem_rec_len - 1;
            sack_size--;
            if (sack_size % 4 == 0)
               printf("\n");
         }
      }
      else
      {
         printf("incomplete SACK \n");
         tcp_opt_size2=-1;
      }
   }
   break;
```

```
            case 8:
               printf("TSOpt ");
               if (rem_rec_len >= TCP_OPT8_LEN - 1)
               {
                  (void)fgetc(fp);
                  printf("%16x ",ReadLong(fp));
                  printf("%16x\n",ReadLong(fp));
                  tcp_opt_size2 = tcp_opt_size2 - 10;
                  rem_rec_len = rem_rec_len - 9;
               }
               else
               {
                  printf("incomplete TSOpt \n");
                  tcp_opt_size2=-1;
               }
               break;
            default:
            //need to add support for length
               printf("unsupported TCP option %i\n",tcp_opt_temp);
               printf("tcp_opt_size2 %i \n",tcp_opt_size2);
               while ((tcp_opt_size2 -1 > 0) && (rem_rec_len > 0))
               {
                  printf("%02x",fgetc(fp));
                  rem_rec_len = rem_rec_len - 1;
                  tcp_opt_size2 = tcp_opt_size2 - 1;
               }
               printf("\n");
               tcp_opt_size2 = 0;
         }
      }
   }
```

```
else if (dumpRec->ip_proto == 17)
{
   dumpRec->udp_src_port = ReadShort(fp);
   dumpRec->udp_dst_port = ReadShort(fp);
   dumpRec->udp_hdr_len = ReadShort(fp);
   dumpRec->udp_chksum = ReadShort(fp);
   printf("UDP: src_port %i dst_port %i hdr_len %i
     chksum %02x\n", dumpRec->udp_src_port,
     dumpRec->udp_dst_port, dumpRec->udp_hdr_len,
     dumpRec->udp_chksum);
   skip_length = dumpRec->rec_len - HEADER_LEN
     - 4*(dumpRec->ip_hdr_len - 5) - 8;
   printf("rec_len %i HEADER_LEN %i ip_hdr_len %i
     skip_length %i skip_length2 %i \n",dumpRec->rec_len,
     HEADER_LEN, dumpRec->ip_hdr_len, skip_length,
     dumpRec->rec_len - HEADER_LEN
     - 4*(dumpRec->ip_hdr_len -5) - 8);
   for (i = 0; i < skip_length; i++)
      printf("%02x",fgetc(fp));
   printf("\n");
   printf("UDP dport %u bytes %u\n",dumpRec->udp_dst_port,
     dumpRec->ip_pkt_len);
}
else
{
   printf("unknown IP protocol %i\n",dumpRec->ip_proto);
   printf("rec_len %i HEADER_LEN %i ip_hdr_len %i\n",
     dumpRec->rec_len, HEADER_LEN, dumpRec->ip_hdr_len);
   for (i = 0; i < dumpRec->rec_len - (HEADER_LEN +
     4*(dumpRec->ip_hdr_len - 5)); i++)
     (void)fgetc(fp);
   }
     printf("\n");

} // End ReadRecord

int main (int argc, char *argv[])
{
   char buffer[256];
   char* filename;
   u_char magic_number[4];
   u_short temp[2];
   u_short pcap_ver_mjr, pcap_ver_min;
   long offset_sec, offset_acc, link_type;
   FILE *fp;
   int i, r, option_idx;
   DumpRecord dumpRec;

   while ((r=getopt_long(argc,argv, "eor:", main_longopt,
     &option_idx)) != -1)
   {
      switch (r)
      {
         case 'e':
            printf("e opt\n");
            break;
         case 'o':
            printf("o opt\n");
            break;
         case 'r':
            if (strlen(optarg) > 100)
            {
```

```
            printf("filename too long\n");
            return 1;
         }
         else
         {
         filename=optarg;
         printf("filename is %s\n", filename);
         }
   }
}

if ( (fp = fopen(filename, "r")) != NULL)
{
   printf("successful in opening %s in read mode\n", filename);
   for (i = 0; i < 4; i++)
      magic_number[i]=fgetc(fp);
   printf("The magic number is %x%x%x%x\n",magic_number[0],
     magic_number[1], magic_number[2],magic_number[3]);
   pcap_ver_mjr = ReadShort_r(fp);
   pcap_ver_min = ReadShort_r(fp);
   printf("Pcap version is %i.%i for file %s\n",pcap_ver_mjr,
     pcap_ver_min,filename);
   offset_sec = ReadLong_r(fp);
   offset_acc = ReadLong_r(fp);
   printf("Local time offset and accuracy are %i sec %i sec\n\n",
     offset_sec,offset_acc);
   snap_len = ReadLong_r(fp);
   link_type = ReadLong_r(fp);
   printf("snap length for the link type %i is %i\n",
     link_type,snap_len);
   while (!feof(fp))
```

```
      {
         ReadRecord(fp,&dumpRec);
      }

   }
   else
   {
      printf("file open failed\n");
      return 1;
   }
   printf("Program ending. closing file\n");
   fclose(fp);
   return 0;
}
```