Master Thesis in Statistics and Data Mining

Mobile Network Traffic Analysis Based on IP Log data

Munezero Parfait

Division of Statistics
Department of Computer and Information Science
Linköping University

Supervisor

Prof. Mattias Villani

Examiner

Anders Nordgaard

Contents

Bibliography **45**

Abstract

The aim of this thesis is to analyze the mobile traffic network in order to provide a deep understanding of the behaviors in the network that could serve as basis for tuning and optimizing the network.

The thesis explores different methods of finding frequent patterns present in the mobile traffic network and propose a probabilistic model to simulate mobile traffic data. The analyzed data was provided by Ericsson, and consists in ordered sequences of the tuples (upload bytes, download bytes, time gap) known as IP bursts (aggregate statistics of IP packets measurements).

SPAM (Sequential Pattern Mining using a Bitmap representation) algorithm is used to extract patterns (such as subsequences of the burst sequences) that appear frequently in the database. Since SPAM doesn't provide information about how likely a pattern is to occur in a sequence, we supplement it with n-gram models in order to describe and better understand the extracted patterns obtained with SPAM. Bigram and Trigram models based on three smoothing methods such as Good-Turing, Laplace, and Jeffrey-Perks are formulated and compared. Comparisons of the different n-gram models reveals that Good-Turing has higher predictive power and assigns the lowest probability to unseen events. SPAM showed that only 8 out of 27 possible events are frequent, and the frequent patterns found are correlated with middle and lower values of the burst's upload and download volumes. Furthermore, the bigram transition matrix computed from Good-Turing estimation suggests that the most frequent patterns obtained with SPAM are periodic.

Finally, the thesis propose a Poisson marked point process as a generative model of the burst data, and use the model to simulate IP burst data for studying the properties of the model and its fit to an actual data set. The Approximate Bayesian Computation (ABC) methodology is used to infer the parameters in the model.

Acknowledgments

I would like to thank several people starting with my supervisors. Thank you Professor Mattias Villani at Linköping University, Wei Li and Anders Söderlund at Ericsson for your assistance and guidance throughout this thesis. The ideas and knowledge you shared with me were essential and helpful. Also, I would like to thank Ericsson, in general, for giving me an opportunity to conduct my thesis project within the company, and most importantly for providing the data and all facilities that were needed. More specifically, I would like to acknowledge efforts made by Leif Jonsson in organizing regular standup meetings in which various instructive ideas were shared. Furthermore, I would like to thank my opponent, Lauren Duff, for the great comments and improvement contributions.

Also, I would like to express my gratitude to the Swedish Institute (SI) that offered me a scholarship for my master's studies. Finally, I would like to thank my family and friends, especially Grace Ntaganda, for their moral support and encouragements.

1 Introduction

1.1 Background

Mobile networks are increasingly important for connecting up people and businesses all over the world. Managing such a big network requires a deep understanding of the internet traffic behaviors, and in order to understand these behaviors, measuring and analyzing the internet traffic are essential.

The Internet Protocol (IP) is one of the communication protocols that ensure transfer of information between devices, such as computers and phones, connected on the internet network. In a network that uses the IP as the communication protocol, connected devices are assigned a unique IP address that serves to identify and locate them on the network, and when a connection is established between two devices, transmitted data (such as email, text file, etc.) are broken into many pieces called packets [5]. Each packet is transmitted with an IP header that contain information about the packet, such as the IP source, the IP destination and the size in byte [5]. In order to analyze the traffic network, telecommunication operators collect regularly IP headers transmitted over their network into a log file.

In the IP header log, packets transferred over the network are stored in sequences ordered by the packet timestamps, and each sequence contains packets from the same IP source, i.e. packets from the same user. These data are very challenging with complex statistical properties and they usually result in huge databases [5]. Aggregating IP headers into objects based on packet inter-arrival times result in another type of traffic data called IP bursts that are less sparse and have simpler structure compared to the raw IP headers. Studying burst data is much easier and can help making inferences about the network behavior. This thesis investigates methods of analyzing a mobile traffic based on burst data provided by Ericsson.

A typical mobile traffic analysis consists in exploring and extracting repetitive patterns that appear in the burst database frequently. This approach is known as pattern mining, which is a branch of data mining dedicated to the discovery

of interesting sequential patterns in large databases [1]. An example of a pattern, in the mobile traffic network perspective, is a sequence of bursts represented by the tuples (upload bytes, download bytes, time gap), where the time gap is the time between two consecutive bursts. Mining such sequential patterns can help, the network providers, in understanding the flow of packets in the network which can serve as basis to improve the quality of services.

Sequential databases can be analyzed and summarized using probability distributions where the probability space consists of the set of all unique events that can appear in the sequences. This approach is well known in the field of speech and natural language processing, where the main task consists in predicting the next word in a sentence given a history of previously observed words[11]. A commonly used predictive model for such data is known as the n-gram model [4, 13]. The main contribution of n-gram models is that they provide an opportunity of computing the probability of a given sequence. In comparison to the pattern mining algorithms, n-gram models have the advantage of providing prediction estimates for the sequences; this can be useful in various ways, such as equipment planning and optimizing the network performance.

Substantial information about network behavior can be gained through simulation. Simulation can be an alternative way of providing data for analysis, and it can help in testing the effects of potential future modifications of network infrastructure. Ideally, one would like a probabilistic generative model, but there has so far been little previous work with such models for network traffic data. One of our aims here is to contributed to the filling of this gap.

1.2 Aim

The aim of this thesis is to provide a deep understanding of mobile network traffic behavior. In order to reach this aim, we explore methods for finding frequent patterns in IP burst data, formulate predictive models for the patterns, and propose simulation as method to study aspects of mobile network traffic.

We explore and extract frequent sequential patterns using SPAM (Sequential Pattern Mining Using an Extended Bitmap Representation) algorithm [2]. However, as SPAM doesn't provide information about how likely a pattern is to occur in a sequence, we supplement it with n-gram models in order to describe the extracted patterns.

Finally, we propose a Poisson Marked point process as a generative model of the data, and use the model to simulate IP burst data for studying the properties of the model and its fit to an actual data set of activity in Ericsson's mobile network.The burst aggregation complicates the likelihood function and we therefore use Approximate Bayesian Computation (ABC) [3] for the inference.

2 Data

The data analyzed in this thesis consist of traffic measurements obtained from Ericsson's mobile network. Data was collected for a large number of users during a busy hour, in which the network is the most congested, and it is usually collected from different network operators (Ericsson's customers) and different markets (i.e. from different geographic areas). One hour of traffic measurement can result in more than 100 GB of traffic log. Two types of traffic logs were collected during the busy hour, namely the IP header logs and the IP burst logs. An IP header contains basic information about packets; it shows the arrival time of the packet, the size in bytes, the direction (upload or download), and the application tag (for example, email, chat, Facebook, etc.). In the IP header log file, packets are ordered based on their arrival times (timestamps). The burst log provides aggregate statistics of the IP header packet data. During a certain transmission period (one hour for our data), IP packets data are aggregated continuously into bursts based on the packets arrival times. Starting from the first packet, a burst is created and all subsequent packets separated by an inter-arrival time less than two seconds are appended to the burst; a new burst will be created only when the time gap between consecutive packets exceed a threshold (two seconds). Bursts are ordered based on the timestamps of the first packets. Since in this thesis we analyze the burst data, we dedicate the next section to the discussion of the burst data in details.

2.1 Burst data

A burst log contains a total of 18 features, but only eight features described in Table 2.1.1, are important for our analysis. These main features carry the following information: the timestamps of the first and the last packet, total number and size of upload packets, total number and size of download packets, the time gap between consecutive bursts, and the duration of the burst.

Table 2.1.1: Main features of the burst data.

Name	Type	Description
Timestamp	string	arrival time of the first packet in the burst
User ID	character	anonymized user ID
Number of upload packets	integer	total number of upload packets in the burst
Number of upload bytes	integer	total volume of upload pakets in the burst
Number of downlink packets	integer	total number of upload packets in the burst
Number of downlink bytes	integer	total volume of upload pakets in the burst
Time Gap	float	time (in sec) between two consecutive bursts
Burst duration	float	time (in sec) a burst lasted

2.2 Data Transformation

Since one burst can contain either both upload and downlink packets or only one of them (leaving the other variable at zero), we use the following transformation to transform the data to the log scale. Moreover, we transformed the time scale in minutes instead of the original scale, which was in seconds. These transformations make the data more amendable for exploratory analysis.

$$f(x) = \begin{cases} ln(x) & if \quad x > 0 \\ 0 & if \quad x = 0 \end{cases}$$

2.3 Data Exploration

In this subsection, we explore the burst data. We visualize the distribution of the main features, as well as the relationships among them.

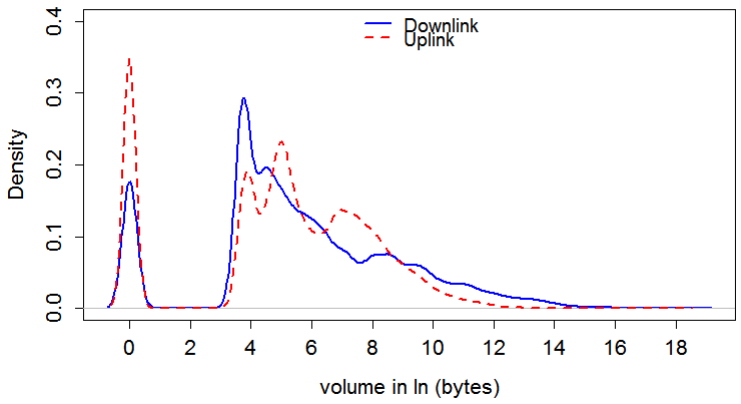

Figure 2.3.1: Distribution of the upload volume (uplink) and download volume (downlink) .

Figure 2.3.1 shows the distributions of the upload volume, download volume. Notice that there is no data between 0 and 3.6 (corresponding to 40 bytes in the original scale); this is due to the fact that a packet volume is bounded between 40 bytes and 1500 bytes. At zero, the upload (dashed line) has a higher density because there are more bursts that include only downlink packets (i.e. when the upload volume = 0, then the burst contains only download packets). Furthermore, we observe much fluctuations when both upload and download volumes are less than 20,000 bytes, where higher peaks appear at :

- upload: 0, 3.9, 5, 6.9, and 7.5 corresponding to the values 0, 50, 148, 992, and 1808 bytes respectively

- download: 0, 3.75, 4.5, and between 8, and 8.6 equivalent to 0, 43, 90, 2981, and 5432 bytes respectively

On the other hand, from Figure 2.3.2 we can see that the time gap and burst duration are exponentially distributed, and some outliers in the time gap can be observed around 4 minutes. Also, we can see that the duration of a burst is most likely to be less than 0.4 minutes, while the time gap can reach even 5 minutes.

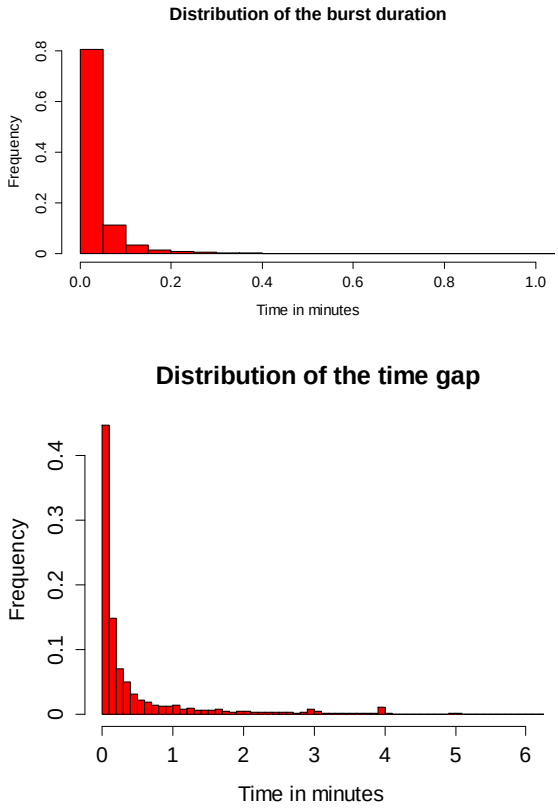

Figure 2.3.2: Distribution the burst duration (top), and the time gap (bottom).

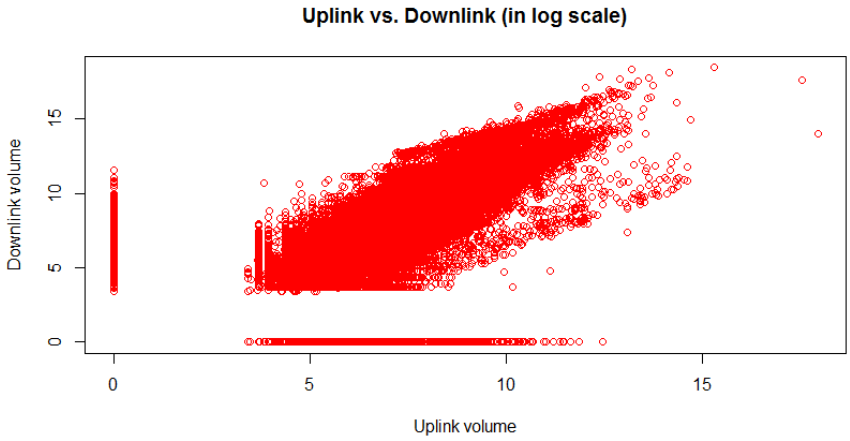

Figure 2.3.3: Upload volume. vs Download volume.

Figure 2.3.3 shows the relationship between the upload and download sizes. We can observe discrete points at different levels of upload volume (uploaded bytes) and download volume (downloaded bytes). These discrete values are observed at 0,

3.6, and 3.9 (on the log scale), which correspond to 0, 40, and 50 bytes . This shows that the upload and download quantities are mixtures of discrete and continuous values.

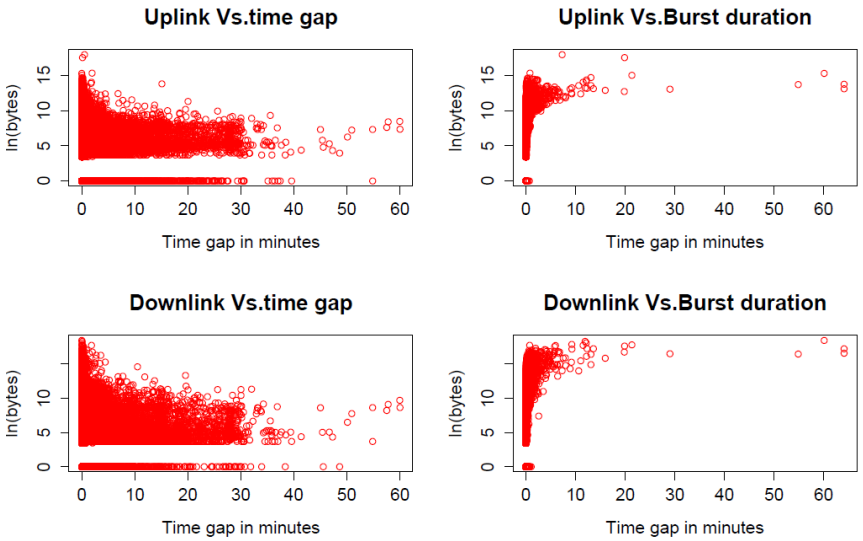

Figure 2.3.4: Relationships among the main features

Also, from Figure 2.3.4, it can be seen that the upload and download quantities decays exponentially as the time gap increases. Another observation to point out is that there are clusters formed in both upload and download quantities when the time gap exceeds 10 minutes. On the other hand, the upload and download sizes increase with the burst duration, but they become too sparse when the duration exceeds 12 minutes and some outliers appear beyond this time.

3 Methods

In this section we discuss different methods used in the analysis of the mobile traffic data.

3.1 Sequence Generation

In the burst log file, an IP burst is marked with the tuple (upload volume, download volume, time gap), where the time gap represents the time between the current burst and the subsequent burst. Generally bursts represent the original events in the burst sequences, however if we consider bursts as events, the sequences will be too sparse and there would be a high risk of running out of memory in the task of pattern mining. So, in order to avoid running out of memory and to reduce the computation time of the pattern mining algorithm, we split the upload volume, download volume, and time gap into three categories: the low, medium and high values (see Figure 3.1.1).

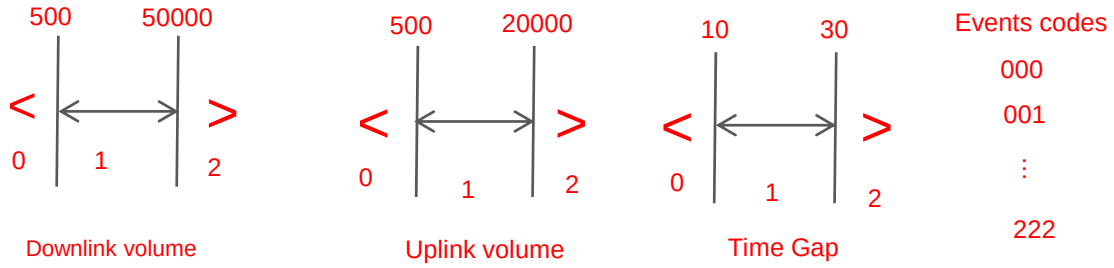

Figure 3.1.1: Event coding

In Figure 3.1.1 we show the defined categories and the coding used to represent events (under the last column of Figure 3.1.1). Events are coded using three digits: the first digit represents the download volume range (0 if the download volume is less than 500 bytes, 1 if the download volume is between 500 and 50000 bytes, and 2 if the download volume is greater than 50000 bytes), the second digit shows the

upload volume range (0 if the upload volume is less than 500 bytes, 1 if the upload volume is between 500 bytes and 20000 bytes, and 2 if the upload volume is greater than 20000 bytes), the last digit shows the time gap range (0 if the time gap is less than 10 sec, 1 if the time gap is between 10 and 30 sec, and 2 if the time gap is greater than 30 sec). For example, if a given burst has the download and upload volumes less than 500 bytes, and the time gap is between 10 and 30 seconds, then the burst is assigned to the event "001". The same procedure applies to the rest of the intervals shown in Figure 3.1.1. In the end, the space formed by the tuples (upload, download ,time gap) is partitioned into 27 discrete unique events.

3.2 SPAM algorithm

In this section, we describe the SPAM algorithm (Sequential Pattern Mining Using A Bitmap Representation), which is an efficient algorithm for mining sequential patterns in large databases [2].

3.2.1 Overview

A database D is a collection of sequences $s_1, s_{2,}...s_n$, where each sequence is a list of k ordered events. A pattern is a segment or a sub-sequence of one or more events that appear at consecutive positions in a given sequence s_i, and the number of events in the pattern determine its length [9]. The problem of pattern mining consists in finding frequent patterns in a large database; the lengths of patterns varies, and generally shorter patterns tend to occur in most of the sequences [2]. Before going into further details of the SPAM algorithm, we define some useful terminology and properties.

- *Supersequence*: A sequence s_i is a supersequence of another sequence s_j if all events in $s_j$occurred in s_i [9].
- *Support*: The support of a pattern is the number of sequences (usually expressed in percentage) in the database D in which it is found. A pattern is deemed frequent if its support exceeds a certain pre-defined minimum support threshold [9].
- *Apriori Property:* The Apriori property sates that all nonempty subsequences of a frequent supersequence are frequent[9]. The idea behind this property is

that if a sequence, say $P_1 = \{\text{"}000 - 001 - 102\text{"}\}$, is not frequent (i.e. it does not satisfy the minimum support threshold), then any event, say $E =$ "222", added to the sequence P_1 cannot occur more often than P_1; thus the resulting supersequence $P_2 = P_1 \cup E$ will not be frequent.

3.2.2 SPAM Description

SPAM is among the family of apriori-based algorithms which proceeds in two subsequent phases. In the first phase, the algorithm scans the database to generate a set of candidate frequent patterns of length $k = 1, 2, \ldots$,denoted by C_k. In the second phase, it scans the database to compute the counts of all patterns in C_k and use the apriori property to prune infrequent patterns in C_k (i.e. patterns that do not fulfill the minimum support threshold). The output consist of sets of frequent k-patterns, which we can denote by L_k. In Algorithm 3.1, we provide the apriori algorithm which is described in details in [1]. The process starts by scanning the database for the first time and returning the set of frequent single event patterns L_1, then C_k, $k = 2, 3, ..$ is generated subsequently by joining patterns in L_{k-1}. The algorithm terminates when no more candidate sets can be generated.

Algorithm 3.1 Apriori algorithm

1. L_1= {1-patterns};
2. for ($k = 2$; $L_{k-1} \neq \varnothing$; $k++$) do begin
3. C_k= apriori-generation from L_{k-1} ; // New candidates
4. for all sequences $s \in D$ do begin
   a) C_s= subset (C_k,s); // Candidates contained in s
   b) for all candidates $c \in C_s$ do
   c) count++;
5. End
6. $L_k = c \in C_k \mid c.count \geqslant minsup$
7. end
8. Return $\cup_k L_k$;

SPAM applies the apriori algorithm and uses the vertical bitmap data represen-

tation: a vertical bitmap represents data as 0,1 bits. Each bitmap represents one sequence, and if an event, e, appears in the sequence at the position i, then the bit for the event e is set to 1; otherwise it is set to 0. This representation makes the process of counting the support very efficient, especially for large databases[2]. However, SPAM is not space-efficient compared to other pattern mining algorithms[2]. Details about SPAM and the vertical bitmap representation can be found in [2].

We selected SPAM, because it is time efficient, and its implementation is available in an open-source data mining library written in JAVA, SPMF (sequential pattern mining framework) [7].

3.3 N-gram models

The n-gram models are well known as language models that attempt to predict the next word in a sentence given a history of previous words[13]. Although, they were originally developed for analyzing text data, n-gram models can be applied to any sequence of events or objects [11]. In the language modeling terminology, an n-gram represents a set of n words grouped together; for instance a bigram ($n = 2$) denotes a set of two words taken at a time, and the vocabulary V is the set of all possible words that can appear in the training set. Throughout this section we use the terminology: sentence and word to refer to a sequence and an event, respectively.

Given a sentence of l words, $S = w_1 w_2 \dots w_l$, an n-gram model estimates the conditional probability of the current word given a set of previously observed words.

$$P(w_n|w_1, \dots, w_{n-1}) = \frac{p(w_1 w_2 \dots w_n)}{p(w_1, \dots, w_{n-1})} \, , n < l \tag{3.3.1}$$

This model reflects the Markov assumption that the next word in the sentence is influenced only by the $n-1$ most recent words; that is, an n-gram model can be thought as a $(n-1)^{th}$ order Markov model, with $n = 2, 3$ (i.e. bigram and trigram respectively) being the most commonly used language models [13]. Let N denote the total number of words in the training set, and $C(w_1 w_2 \dots w_n)$ the frequency of an n-gram $w_1 w_2 \dots w_n$ in the training set. The obvious way of estimating the probabilities in (3.3.1) would be using the maximum likelihood estimator (MLE) in

3.3.2, which is proven to be the observed relative frequencies of the sequences [13].

$$\begin{aligned} P_{MLE}(w_1 w_2 \dots w_n) &= \frac{C(w_1 w_2 \dots w_n)}{N} \\ P_{MLE}(w_n | w_1, \dots, w_{n-1}) &= \frac{C(w_1 w_2 \dots w_n)}{C(w_1, \dots, w_{n-1})} \end{aligned} \tag{3.3.2}$$

Generally, all words in the vocabulary do not appear in the training set; some are more frequent, others are less frequent or are never seen. If a sentence contains a word that was never seen before, the MLE estimates assigns zero probability to the sentence. This makes the MLE unsuitable for this kind of modeling. So in order to alleviate the MLE problem, a number of smoothing methods that produce smoothed probabilities (i.e. zero free probability density) have been developed, and a survey of some of them can be found in[4]. These methods decrease the probability of the n-grams encountered in the training set in order to reserve some probability mass to unseen n-grams.

3.3.1 Laplace law and Jeffrey-Perks law

The Laplace law, expressed in Equation 3.3.3 below, is commonly referred to as the *add one* method, and assumes that each word in the vocabulary occurred once more than it appeared in the training set. This is equivalent to assume that, a priori, every n-gram has the same probability of being observed (i.e. it is the Bayes estimator obtained when one applies a uniform distribution over the n-grams)[13].

$$P_{lap}(w_1 w_2 \dots w_n) = \frac{C(w_1 w_2 \dots w_n) + 1}{N + v} \tag{3.3.3}$$

where v is the size of the vocabulary (i.e. the number of all possible unique words), and N the total number of words in the training set.

An alternative smoothing method, closely related to the Laplace law, consists in adding a small positive number, α, to each n-gram count as in Equation 3.3.4. One can use any number between 0 and 1, but the Jeffrey-Perks law suggests $\alpha = 0.5$, which is interpreted as the interpolation between the MLE and Laplace estimates[13]. In contrast to the Laplace law, the method of adding alpha assigns a

smaller probability mass to unseen words [13].

$$P_{alpha}(w_1 w_2 \dots w_n) = \frac{C(w_1 w_2 \dots w_n) + \alpha}{N + \alpha v} \tag{3.3.4}$$

3.3.2 Good-Turing smoothing method

The Good-Turing smoothing is one of the more advanced smoothing methods, which assumes that the n-grams are independent and that the number of times an n-gram occurs has a binomial distribution. If we define, N_r as the number of n-grams that occurred exactly r times in the training set, and N the total number of observed words, the Good-Turing treats an n-gram that occurred r times as if it appeared $r+1$ times. Good-Turing uses adjusted frequencies $r^* = (r+1)\frac{N_{r+1}}{N_r}$ to compute the probability of a given n-gram according to

$$P_{GT}(w_1 w_2 \dots w_n) = \begin{cases} \dfrac{r^*}{N} & if\ r > 0 \\ \dfrac{1 - \sum_{r=1}^{\infty} N_0 \dfrac{r^*}{N}}{N_0} \approx \dfrac{N_1}{N_0 N} & if\ r = 0 \end{cases} \tag{3.3.5}$$

Using the observed values of N_r to compute the adjusted frequencies (r^*) can lead to inaccurate results, especially for higher values of r, since they are generally too sparse (sometimes no n-gram appeared $r+1$ times, i.e.$N_{r+1} = 0$). One solution to this problem is to estimate r^* using the observed N_r only for small values of r (say $r < 10$); otherwise for higher values of r, use smoothed values of N_r obtained by fitting a smoothing curve through the observed values of r and N_r [13]. It is suggested in [13] to use the simple linear regression function: $\ln(N_r) = a + b \times \ln(r)$ (where a and $b < 0$ are constants to be estimated) as the smoothing curve.

3.3.3 Perplexity

The perplexity is a standard measure of performance of a language model, and it can be used to compare different models. If the test data T is composed of sequences$(s_1, s_2, \cdots s_l)$ the perplexity on the test data is defined as:

$$pp = 2^{-\frac{1}{N_T}\sum_{i=1}^{l} log_2 p(s_i)},$$

where N_T is the total number of words in the test set, and $p(s_i)$ the probability of a sequence s_i in the test data. A model with small perplexity has a higher predictive performance.

3.4 A marked point process for IP burst data

In some situations one may have access only to a dataset that comes from transforming or aggregating unobserved raw data. If one knows some information on the raw data properties and how the transformation was done, then one can formulate a probabilistic model that simulates the unobserved raw data, and apply the same transformation in the simulation processes in order to make both the simulated and observed data comparable. The observed secondary data can then be used to learn the parameters in the underlying data generating process.

Our generative model follows the same procedure since the observed data was obtained by aggregating IP packet data. We simulate unobserved arrival times as well as the size (number of bytes) of individual IP packets, and then aggregate the packets into bursts, which are our observed data. We use the data at the burst level to learn the parameters in the model. However, the process of aggregating packets into bursts complicates the derivation of the likelihood function, which makes it hard to apply commonly used optimization techniques for estimating the optimal model parameters. However, it is still possible to learn the parameters without formulating the likelihood function, by using the Approximate Bayesian Computation (ABC) methodology [14]. In the following sections, we discuss in full details the point process and the idea behind our generative model, as well as the ABC inference scheme .

3.4.1 Point process

Internet data can be regarded as arrival times of packets following a point process[5], and if each time point is appended with the size of the packet the process becomes a marked point process[12].

A point process is a method of allocating random points (the points are generally

the times of observing some particular events) to an interval on the real line [12]; the interval is usually defined on the positive half real line$(0, \infty)$. Let $t_k,\ k = 1, 2, ..., n$ denote the arrival time of the k^{th} event. Then t_k can be expressed as the sum of the lengths of the k intervals $(t_0, t_1], (t_1, t_2], \ldots, (t_{k-1}, t_k]$, defined from the starting time $t_0 = 0$. There are various instances of point processes, however in this thesis, we consider the Poisson point process, which is the most commonly used point process.

A Poisson point process assumes that the length of the intervals $(t_0, t_1], (t_1, t_2], \ldots, (t_{k-1}, t_k]$ are independent and identically exponentially distributed. i.e $(t_k - t_{k-1}) \sim^{iid} exp(\lambda)$ [12]. From this property, we can simulate the point process on a given interval $(0, T)$ by applying the inverse CDF sampling method [8], where the inverse transformation of the exponential distribution function with the parameter λ is expressed as:

$$T = \frac{-ln(u)}{\lambda}, \qquad 0 < u < 1 \tag{3.4.1}$$

Referring to Equation 3.4.1 , the simulation of the point process follows Algorithm 3.2

Algorithm 3.2 Point process simulation

start from $t_0 = 0$

1. Generate a random number u from the uniform distribution U (0,1)
2. compute $t_k = t_{k-1} - ln(u)/\lambda$
3. repeat 1 to 2 until t_k exceeds the observation time T

3.4.2 A generative model for IP bursts

Our simulation model generates data for one user at a time. For reasons of simplicity, we assume that the users are independent and the data generated by different users have the same distribution (i.e. identically distributed). Furthermore, there are two aspects to consider: the size of a packet is restricted to the interval $[40, 1500]$, and its distribution is a mixture of discrete and continuous values (as noted in Figure 2.3.3). Since the packet size is always positive, and given that we would like to incorporate the correlation structure of the packets in the model, the lognormal distribution seems to be a suitable candidate for modeling the continuous values of the packet size.

The simulation procedure for the data generated by one user is summarized in the

following five steps, where the first three steps simulate the intermediate unobserved IP packet data, and the last two steps aggregate these packets into the packet bursts, producing our observed data. The steps are as follow:

1. Simulate a sequence of packet arrival times, $t_1, t_2, \ldots, t_n$ from the Poisson point process on $(0, T]$, where T is 60 minutes in our case.

2. At each time t_i a packet is either uploaded ($u_i = 1$) or downloaded ($u_i = 0$). Simulate the activity indicators $u_1, u_2, \ldots, u_n$ from a Markov chain with a transition matrix $\begin{bmatrix} \pi_{00} & (1-\pi_{00}) \\ (1-\pi_{11}) & \pi_{11} \end{bmatrix}$, where π_{00} denotes the probability of downloading given that the user was previously downloading (π_{11} expresses a similar scenario for the upload state).

3. Let y_i denote the log of the upload/download volume at time t_i, and let y_u be a vector collecting all y_i where $u_i = 1$, i.e. y_u is a vector of all upload quantities. Similarly, y_d represents the vector of all download quantities. Simulate, y_u and y_d independently from the mixture model in 3.4.2:

$$\begin{aligned} y_u &\sim P_{u1} * I_{40} + P_{u2*} N_{\|u\|}(\mu_u, \Sigma_u) + P_{u3} * I_{50} \\ y_d &\sim P_{d1} * I_{40} + P_{d2*} N_{\|d\|}(\mu_d, \Sigma_d) + P_{d3} * I_{50} \end{aligned} \tag{3.4.2}$$

where $I_c(x) = \begin{cases} 1 & if\ x = c \\ 0 & otherwise \end{cases}$, and $N_{\|.\|}(.)$ denotes a multivariate normal distribution. p_{ui}, $i = 1, 2, 3$ (alternatively p_{di} for download quantities) denotes the probabilities of drawing y_u (alternatively y_d) from: $I_{40}(x)$, the multivariate normal distribution, and $I_{50}(x)$ respectively. The mixture probabilities sum to one, i.e. $\sum_{i=1}^{3} p_{ui} = 1$ and $\sum_{i=1}^{3} p_{di} = 1$. Also, we would like to define the covariance matrices in the multivariate normal distribution, in such a way that the correlation between any pairs in the vector y_u or y_d decays towards zero as the inter-arrival time between them increases. One possible way is to use the exponential correlation model [6], expressed as $\Sigma_u = \sigma_u^2 \exp(-\phi_u |t_j - t_k|)$ for upload packets, and $\Sigma_d = \sigma_d^2 \exp(-\phi_d |t_j - t_k|)$, for download packets. In order to bring y_u and y_d, computed from the multivariate normal distribution, within the proper limits of a packet size we use the transformation:

$$f(y) = ln(40) + \frac{[y - min(y)] * [ln(1500) - ln(40)]}{[max(y) - min(y)]}$$

4. Return the download and upload vectors y_u and y_d in the exponential scale (i.e $exp(y_u)$ and $exp(y_d)$), and aggregate the packets into sequences of bursts $b_1, b_2, \ldots, b_k$. Starting from the first packet, append the subsequent packet to the current packet until the inter-arrival time between both packets exceeds 2 seconds. A new burst is created when the threshold is exceeded and it begins from the first packet arriving right after the end of the previous burst, and the process continues until the observation time is reached.

5. Return the sequences of burst volumes (z_{uj}, z_{dj}) consisting of pairs of the sums of all upload quantities and download quantities belonging to the same burst.

The proposed model has a total of 13 hyper-parametersλ, μ_u, μ_d, σ_u, σ_d, ϕ_d, ϕ_u,P_{u1},P_{u2}, P_{d1}, P_{d2}, π_{00}, and π_{11}. In order to simplify the notations, in the next section we refer to all these parameters jointly as one multidimensional parameter θ.

3.4.3 Approximate Bayesian Computation (ABC)

To avoid setting up the complicated likelihood function for the model of the burst data, we use ABC techniques[14] to approximate the posterior distribution of θ given the observed burst data. The key idea underlying ABC is that data simulated from the model with θ-values that have large posterior density tend to resemble the observed data. To measure the similarity between data simulated from the model and the observed data, ABC relies on fairly low-dimensional summary statistics $S(y_i)$ of the data. There are by now a large number of ABC algorithms available, but in this thesis we use the ABC-MCMC algorithm [14] and the standard ABC rejection algorithm [15] and compare the results.

Algorithm 3.3 summarizes the standard ABC rejection algorithm. A parameter θ' is drawn from the prior distribution $\pi(.)$, and it is used to simulate a dataset y from which a summary statistics $S(y)$ is computed. The process is repeated n times, and the simulated θ_i $i = 1, 2, \ldots, n$ that yielded the summary statistics $S(y_i)$ closest to the target summary statistics $S(y_0)$ (i.e summary statistics of the observed data) are retained to form the posterior distribution of the model parameters. $S(y_i)$ is deemed close to $S(y_0)$ if the euclidean distance between the two summary statistics, $d(S(y_i), S(y_0))$, does not exceed a certain predefined threshold known as the tolerance level ϵ [15].

Algorithm 3.3 ABC rejection sampler

for $i = 1$ to N do
repeat
    Draw $\theta^{'}$ from the prior distribution $\pi(.)$
    Generate y from the simulation model $f(.|\theta^{'})$
until $d(s(y), s(y_0) \leq \epsilon$
    set $\theta_i = \theta^{'}$
end for

Using simulations from the prior distribution is inefficient because if the prior is wrong it could result in low posterior probability regions [14]. The ABC-MCMC algorithm in Algorithm 3.4 extends the Algorithm 3.3 in order to avoid using simulations from the prior distribution.

Algorithm 3.4 ABC-MCMC sampler

Use Algorithm 3.3 to initial realization $(\theta^{(0)}, y^{(0)})$
for $t = 1$ to N **do**
    Generate $\theta^{'}$ from a Markov Kernel $q(.|\theta^{(t-1)}$
    Generate $y^{'}$the simulation model $f(.|\theta^{'})$
    Generate u from$U(0, 1)$
if $u \leq \frac{\pi(\theta^{'})q(\theta^{(t-1)}|\theta^{'})}{\pi(\theta^{(t-1)})q(\theta^{'}|\theta^{(t-1)})}$ and $d(s(y^{'}), s(y_0)) \leq \epsilon$ **then**
    set $(\theta^{(t)}, y^{(t)}) = (\theta^{'}, y^{'})$
else $(\theta^{(t)}, y^{(t)}) = (\theta^{(t-1)}, y^{(t-1)})$
end if
end for

Both Algorithm 3.3 and Algorithm 3.4 are implemented in the EasyABC R-package [10]. We use the package EasyABC to construct the posterior distribution of θ.

The posterior approximation obtained with the rejection algorithm can be improved by using regression techniques to reduce the discrepancy between the accepted summary statistics $S(y_i)$ and the target statistics $S(y_0)$. These techniques are implemented in the ABC package. The ABC package implements two regression methods to adjust the posterior approximates. One method is the local linear regression and the other is the neural network method. The local linear method, which is used in this thesis, adjusts the posterior approximation of θ through a linear regression function and it requires specifying the transformations that should be applied to the parameters before adjusting them. Since the probability is always between 0 and 1 the parameters p_{d1}, p_{d2}, p_{u1}, p_{u2}, p_{00}, and p_{11} are transformed using the logit transformation with 0 as the lower bound and 1 as the upper bound, and

the rest of the parameters are transformed to the log scale since they are positive.

3.4.3.1 Choice of summary statistics

We set up a vector of 11 summary statistics: the total number of bursts, the ratio of the upload packets and download packets, the rate of bursts consisting of only upload or download packets (burst with zero upload volume or zero download volume), and the mean and standard deviation of the following variables : upload volume per packet and download volume per packet, time gap, and burst duration. We computed the mean upload (and download) volume per packet by averaging the burst's upload (and download) volume divided by the number of upload (and download) packets in the burst.

Since the simulation of the whole dataset can take a long time, it is preferable to simulate data for one user at a time (users are assumed to be independent). It is, therefore, required to estimate the expected value of the target statistics (the observed statistics used in ABC procedure) relative to the data of one user. We estimate this expected value by drawing 10000 random samples of users from the observed data and computing the average. This average, in the long term, converges approximately to the expected summary statistics of the data generated by one user (by the law of large numbers).

3.4.3.2 Prior distribution

Since we do not have much information about the hyper-parameters, we set up uninformative priors on θ. We set a uniform prior $U(0, 0.5)$ on the parameters P_{ui}, P_{di},π_{00} and π_{11} . We restricted these probabilities from being larger than 0.5 because beyond this value the Markov chain tends to stay in one state, resulting in a higher rate of bursts with zero upload/download packets. Also with a higher value of P_{u1} or P_{d1} too many discrete values are generated. For the rest of the parameters, the prior is set to $U(40, 1500)$ for the mean upload and download per packet, $U(0, 1)$ for both decay factors in the exponential correlation function. The priors for the standard deviations for upload and download size per packet were set to $U(0, 80)$ and $U(0, 200)$ respectively. Finally, we set $U(1, 100)$ as the prior for the rate of the packet arrival times (the parameter λ in the point process).

4 Results

In this section we present the results obtained from our tasks including the frequent pattern mining, n-gram models, and the probabilistic model of simulating the traffic data. The section on pattern mining combines results obtained from the SPAM algorithm and the results obtained from the n-gram models.

4.1 Pattern Mining

4.1.1 Pattern Exploration

The training set has 1819 sequences with a total of 149406 events. Figure 4.1.1 shows the distribution of the sequence lengths, which consist of the number of events included in the sequences. On average a sequence has around 82 events; the minimum observed number of events in a sequence being one and the maximum being 731. However, it is more likely to observe a sequence with less than 200 events.

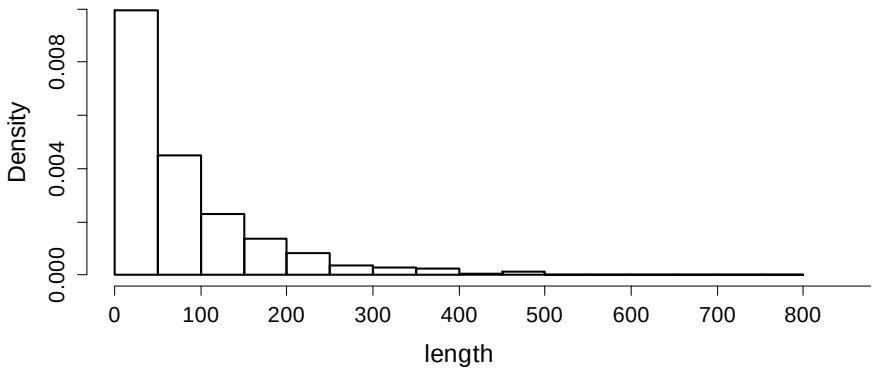

Figure 4.1.1: Distribution of the length of the sequences.

Four events: “000”, “110”, “111”, “001” , are the most prevalent (see Figure 4.1.2). Recall that the first digit in the event coding shows the range of download volume,

the second shows the range of upload volume, and the third represents the time gap range.

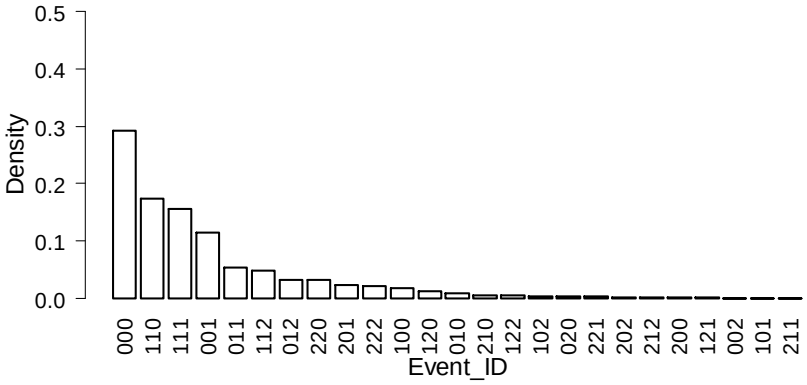

Figure 4.1.2: Distribution of events.

4.1.2 Minimum support threshold selection

SPAM requires setting the minimum support threshold; that is the minimum frequency a pattern should fulfill to be considered as frequent. We experimented with the following thresholds in a small database (26 users) : 25%, 30%, 40%, 50%, 60%, 70%. Results can be seen in Table 4.1.1.

Table 4.1.1: Results from SPAM with different minimun support thresholds.

Min_Sup	**Frequent single events**	**frequent patterns**	**Time (msec)**
70%	000, 001, 002, 110,112	91	393
60%	000, 001, 002, 110, 111, 112	690	1074
50%	000, 001, 002, 110 ,111, 112	7116	7671
40%	000, 001, 002,100,110, 111, 010, 112	135866	113282
30%	000, 001, 002,100,110, 111, 010, 112 ,220,120	6392898	4112310
25%	Low memory		

With 25% the algorithm crashed because of memory insufficiency, with 30% it took a long time to complete (around one hour), and with 70% it discarded too many events resulting in few frequent patterns. A threshold of 40% strikes a good balance between computing time and finding enough frequent patterns.

4.1.3 Frequent patterns

With 40% minimum threshold, 1242474 frequent patterns (most repetitive subsequences) are observed in the training set. The longest pattern has 17 events, and it is observed in sequences generated by 800 different users. This implies that 17 sets of frequent patterns (each containing patterns of the same length) were returned. In Figure 4.1.3, we show the absolute support (support expressed as counts: number of users in which the pattern occurred) against the length of all patterns in each frequent set, and we summarize these sets in Table 4.1.2 .

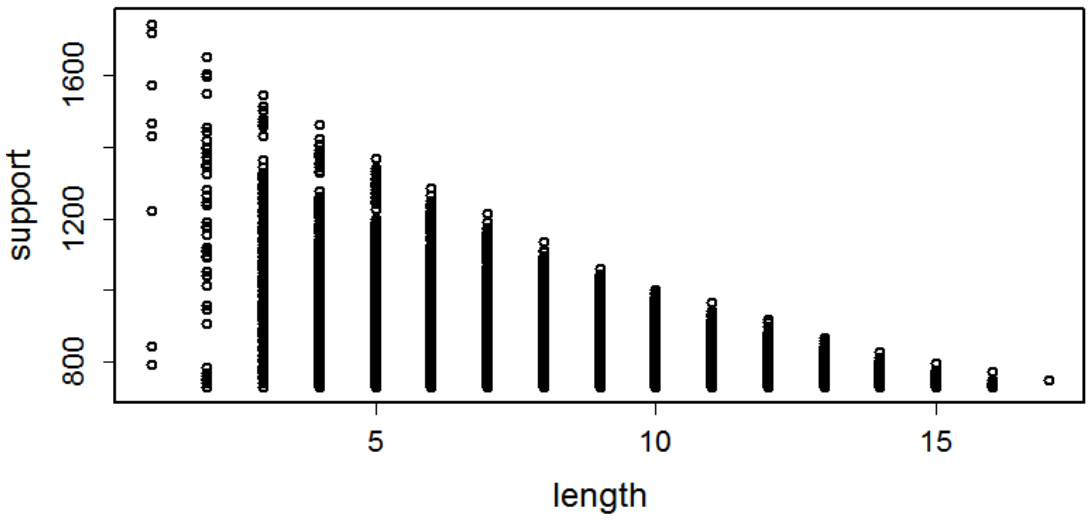

Figure 4.1.3: Support vs. length of frequent patterns.

Obviously the support decreases as the pattern length increases. Out of 27 unique events only eight, namely: “000”, “001”, “002”, “100”, “110”, “111”, “010”, “112” are frequent, and since SPAM applies the apriory property (which states that only supersequences of frequent patterns are frequent) the rest of frequent patterns are combinations of these eight events. Also, we can observe that the support is more scattered for shorter patterns.

For each returned frequent set $L_{k,}, k = 1, \ldots, 17$ (set of equal-length frequent patterns), we extracted the pattern with the highest support and aggregated the results in the second column of the Table 4.1.2. In the third column, are the number of unique events in the set; the last column shows events (among the eight frequent unique events) that never appeared in the patterns belonging to the set.

Table 4.1.2: Summary of the frequent sets.

Freq_set (L_k)	Patterns with the highest support in L_k	Unique events	Lost events
1	000	8	-
2	002-000	8	-
3	002-002-000	7	100
4	002-002-002-000	6	100, 010
5	002-002-002-002-000	6	100, 010
6	002-002-002-002-002-000	6	100, 010
7	002-002-002-002-002-002-000	6	100, 010
8	002-002-002-002-002-002-002-000	6	100, 010
9	002-002-002-002-002-002-002-002-000	6	100, 010
10	000-000-000-000-000-000-000-000-000-000	6	100, 010
11	000-000-000-000-000-000-000-000-000-000-000	6	100, 010
12	000-000-000-000-000-000-000-000-000-000-000-000	4	100, 010, 112, 111
13	000-000-000-000-000-000-000-000-000-000-000-000-000	4	100, 010, 112, 111
14	000-000-000-000-000-000-000-000-000-000-000-000-000-000	4	100, 010, 112, 111
15	000-000-000-000-000-000-000-000-000-000-000-000-000-000-000	3	100, 010, 112, 111, 110
16	000-000-000-000-000-000-000-000-000-000-000-000-000-000-000-000	2	100, 010, 112, 111, 110, 001
17	000-000-000-000-000-000-000-000-000-000-000-000-000-000-000-000-000	1	100, 010, 112, 111, 110, 001, 002

Looking at the last column of the Table 4.1.2, we observe that the event “100” was lost in the frequent set L_3, which means that all frequent patterns containing more than 3 events do not contain the event “100”. From L_4, the event “010” never occurred, and only 6 unique events are repetitive in patterns of length four, and we observe the same scenario in subsequent frequent sets L_5 up to L_{11}. In L_{12}, only four unique events persisted, and the last frequent set has only one prevailing event, which is “000”.

From these observations, we can infer that important frequent sets are L_3,L_{11}, L_{14}, L_{15}, L_{16}, and L_{17} because all patterns belonging to other frequent sets are included in the former sets. Moreover, from the second column, we note that the most frequent patterns in all L_k are included in the following two events: "002-002-002-002-002-002-002-002-000", and "000-000-000-000-000-000-000-000-000-000-

000-000-000-000-000-000-000". In order to understand these patterns we plotted the distribution of applications tagged with the events “000” and “002”.

Figure 4.1.4 presents these distributions, and we can observe that “chat”, and the unlabeled other applications (categorized as “other”) have a sounding predominance.

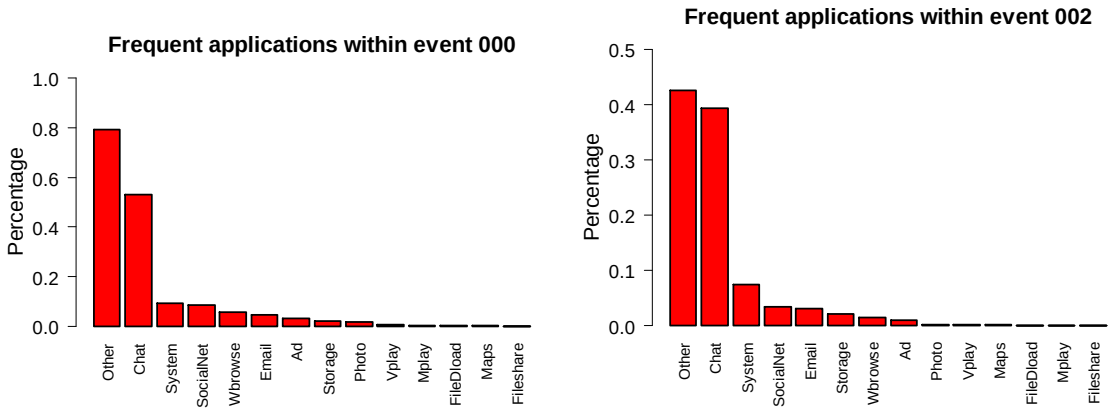

Figure 4.1.4: Frequent applications within the events “000” and “002”.

4.1.4 n-gram models

In this section, we present the results obtained from training different n-gram models; the training set is the same as the one we used in the pattern mining task ; i.e sequences generated by 1819 users containing a total of 149406 events. We evaluate the predictive performance of the models on a test set that has 2686 events. We formulated bigram and trigram models using the Good-Turing smoothing (GT), Laplace law (LAP) and Jeffrey-Perks law (ALPHA).

Table 4.1.3: Perplexity for different n-gram models.

Model	Unseen event Prob	Perplexity
BigramGT	2.741799e-06	28.47346
BigramLAP	6.665334e-06	28.73062
BigramALPHA	3.339623e-06	28.81789
TrigramGT	6.902189e-07	27.69393
TrigramLAP	6.059541e-06	26.50267
TrigramALPHA	3.180328e-06	28.4899

Table 4.1.3 displays the aggregated perplexity for the different models in the test set. From the second column of Table 4.1.3, we observe that the lowest probability

allocated to previously unseen events is obtained by Good-Turing, while the highest probability is allocated by the Laplace smoothing method. In general, all smoothing methods have similar perplexities, and the bigram models have slightly higher perplexities compared to the trigram models. Also, we note that Good-Turing has the lowest perplexity in the bigram models, while in the trigram models Laplace outperforms Good-Turing slightly. On the other hand, the method of Jeffrey-Perks produced more or less the same perplexity values for bigram and trigram models.

In Figure 4.1.5, we show the heatmap plot for the bigram transition matrix obtained by applying Good-Turing smoothing (the transition matrix for the trigram models have too many events to fit in one plot). In this plot, lighter color represent high values, and dark colors maps lower values.

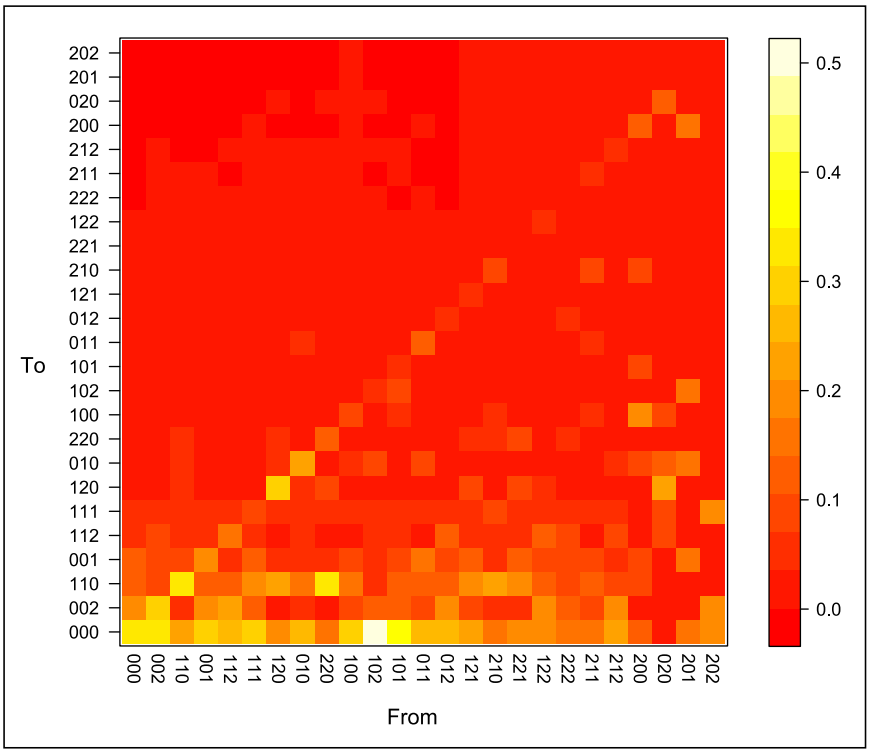

Figure 4.1.5: Bigram model using Good-Turring smoothing.

Figure 4.1.5, shows that various events have higher probability of transitioning into event "000", especially the event "102" where $Pr(\text{“102”} \rightarrow \text{“000”}) = 0.5$. Other forthcoming events that have high probabilities are "110", and "002". Another important aspect to point out is that many events, especially the first six events ("000", "002", "110", "001", "120", "010") tend to be periodic; that is they have a high probability of returning to themselves after a certain number of transitions. Finally, we

can observe that the last three events (“020”, “201”, “202”) are inaccessible from any other event.

4.2 Marked point process

In this section we present results obtained from the generative marked point process model and the ABC inference. We draw 11000 parameters from the prior distributions and launch simulations based on those parameters. In the parameter rejection step, we applied the local linear rejection method (implemented in the ABC package) with 0.05 as the tolerance threshold.The resulting posterior distribution summary is presented in Table 4.2.1.

Table 4.2.1: Posterior summary

	p_{d1}	p_{d2}	p_{u1}	p_{u2}	p_{00}	p_{11}	μ_U	μ_D	ϕ_U	ϕ_D	σ_U	σ_D	λ
Min	0.11	0.55	0.04	0.14	0.00	0.01	0.91	6.92	0.62	0.19	0.00	4.65	47.48
Mean	0.33	0.56	0.34	0.36	0.03	0.02	788.03	837.26	0.91	0.79	41.84	116.80	50.30
Mode	0.32	0.56	0.17	0.31	0.001	0.02	169.18	270.73	0.93	0.79	8.01	148.21	50.48
Max	0.54	0.57	0.70	0.85	0.57	0.05	3338.9	3282.53	0.99	0.97	222.78	188.31	52.99

In Table 4.2.1, p_{d1} and p_{d2} (or p_{u1}, p_{u2} for upload quantities) denote the first two probability parameters in the mixture model 3.4.2. Since the probabilities in the mixture model sum to 1 then the third probability parameter p_{d3} is computed as $(1 - (p_{d1} + p_{d2}))$. p_{00} and p_{11} represent the probability of download and upload in the Markov chain transition matrix respectively.μ_U and σ_U (or μ_D and σ_D for dowload quantities) represent the mean and standard deviation of the upload size per packet. The summary statistics presented in Table 4.2.1 are summaries of 550 parameters draws that were accepted in the ABC procedure. Their distributions are shown in Figure 4.2.1, where we can see the estimated posterior for some parameters. The parameters: λ, p_{d1}, p_{u2},ϕ_U,ϕ_D and p_{11} have distributions close to the normal distribution. Also, we can see, from the third plot in the first raw, that the uniform prior on p_{u1} did not convey much information on its posterior.

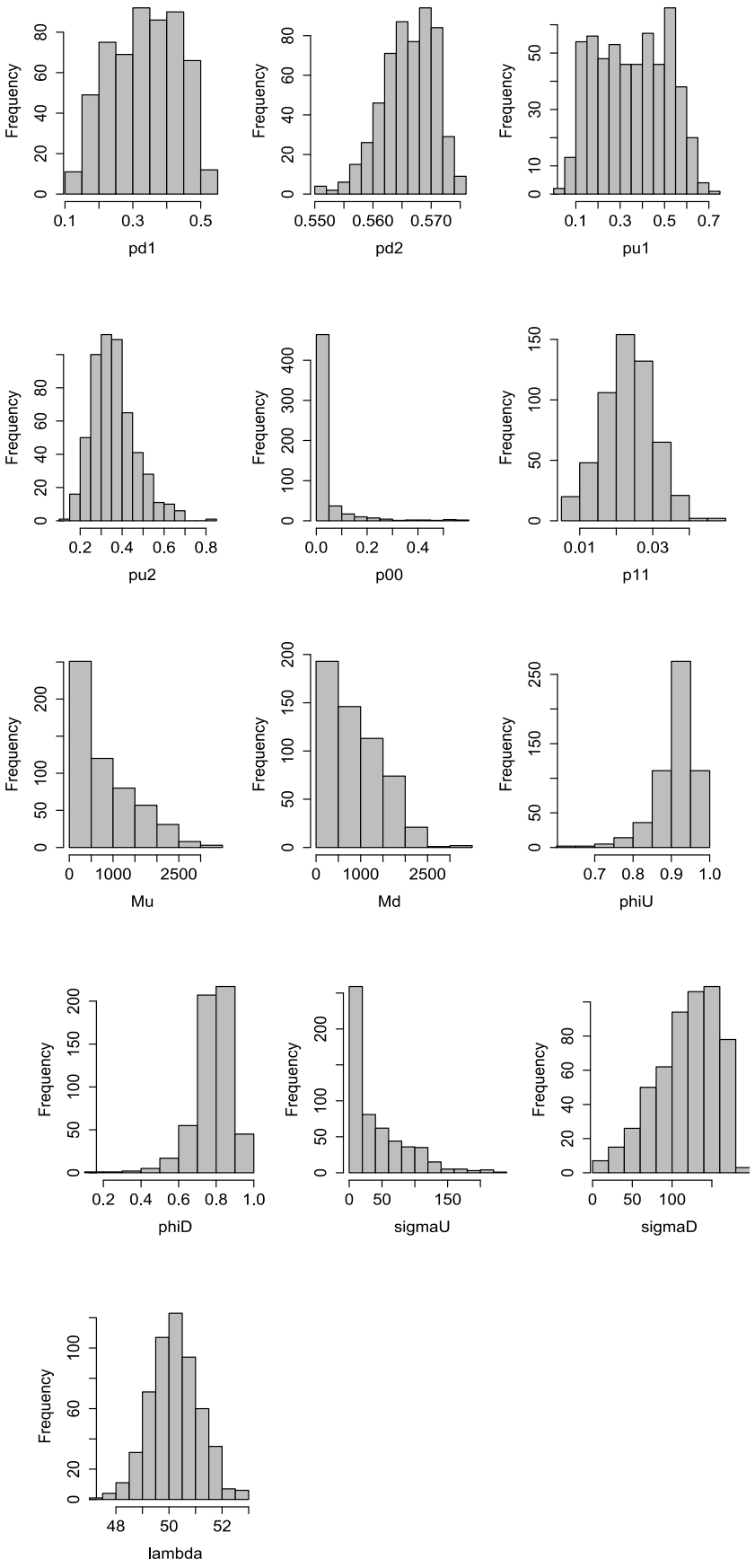

Figure 4.2.1: Posterior distribution

We used the mode vector in Table 4.2.1 to generate data for 100 users and compared the simulated data with the data generated by 100 randomly-selected observed users. In Figure 4.2.2, we present the distribution of upload and download size, time gap, and burst duration for both simulated and observed data. Since within a burst some values of upload size or download size are either strictly positive or zero, we transform a value to the log scale if it is greater than zero and leave zeros as they are (see the transformation in data section).

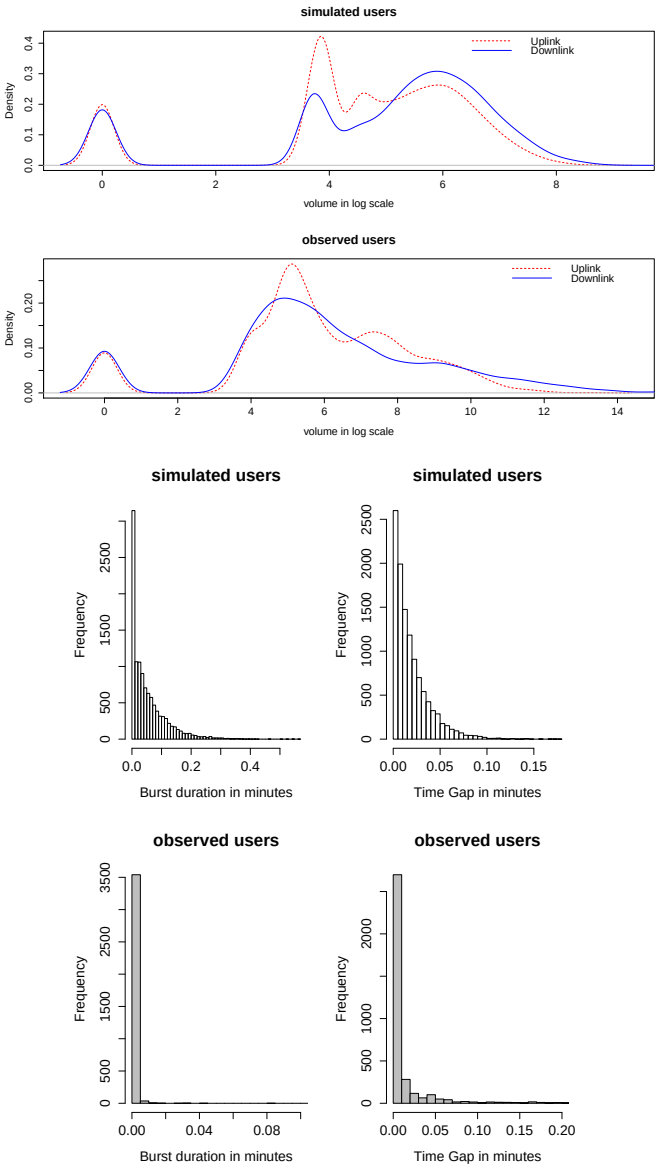

Figure 4.2.2: Distribution of the main burst variables for simulated data and observed data

Comparing the distribution plots for both simulated and observed data, it can be seen in Figure 4.2.2 that the distribution of observed upload has peaks around 4, 5, and 6; similarly the simulated upload has peaks around the same values but the shape of the distribution is different and the peak at 4 is too high. Also, the peak around 6 is much higher in the simulated data compared to the observed data . In addition to that, the distribution of time gap and burst duration decays slowly in the simulated data compared to the decay in the observed data.

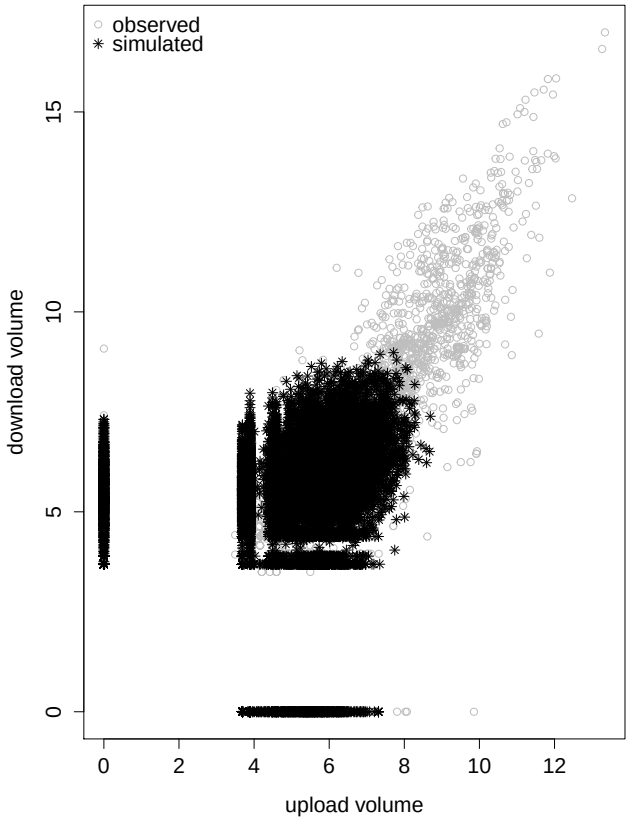

Figure 4.2.3: upload volume and download volume scatter plot for simulated and observed data overlaid

However, from Figure 4.2.3, which presents the overlaid scatter plot of upload size against download size in the log scale for both the observed and simulated data, we notice that the simulation model has captured the behaviors of low levels of upload and download packet sizes. The simulated points have the same shape as the observed points when the log of both upload and download are less than 10.

Since a packet size is limited between 40 and 1500 bytes, a burst should aggregate a large number of packets in order to reach the extreme values of the observed burst upload and download packet sizes. Thus, the model failed to simulate a high number of packets (we discuss the reasons of this failure in the discussion section).

4.3 Discussion

4.3.1 n-gram models

From Table 4.1.3, the Laplace method seems to perform well, but it actually overestimates the probabilities because it allocates higher probability to unseen events. The Jeffrey-Perks method allocates lower probability mass to unseen events compared to the Laplace estimator, but yields slightly higher perplexity. On the other hand, the Good-Turing estimator assigns to unseen events less than half of the probability allocated by the Laplace estimator, and it has a low perplexity in both bigram and trigram models. Overall, the estimates produced by the Good-Turing estimator therefore seems preferable.

Trigram models produced slightly lower perplexity compared to bigram models, which means that they have higher predictive power. However, there is a higher risk of experiencing numerical problems with higher order n-gram models, because of the lengths of the sequences. Some sequences have more than 500 events, and since the estimated n-gram probabilities become too small, the probability of these long sequences result in smaller numbers than the computer can store in memory. This problem occurred only for n-gram models of order higher than two because the number of unseen events becomes larger as the n-gram order increases and the probability assigned to such events is usually too small.

In order to avoid this underflow problem, we have split long sequences into various pieces that have at most 200 events, and regarded the pieces as separate sequences. Doing this, we avoided infinite values that otherwise would occur in the perplexity of trigram models. Hence, for the accuracy reasons we argue that bigram models, for this kind of data, are better than trigram models, even though the latter have higher predictive power.

4.3.2 Pattern Mining

With 40% minimum support threshold, we observed that only 8 events (“000”, “001”, “002”, “100”, “110”, “111”, “010”, “112”) are frequent. The events “010” and “100” occurred only in shorter patterns, whereas, “000”, “002”, “110”, “111”, and “112” prevailed in longer patterns.

In various frequent patterns, we encountered all interval ranges of time gap (0, 1, and 2), but we never observed higher levels of upload and download sizes. Thus,

we can infer that frequent patterns do not correlate with the time gap, but they are more related to the burst size levels. They are correlated with middle and lower ranges of upload and download sizes. Shorter frequent patterns tend to include events with middle level of upload volume (corresponding to the category 1 in the event coding) and lower level of download volume (corresponding to the category 0 in the event coding), however when both upload and download are in the same range, the corresponding events tend to persist in longer patterns. This can be noted from the fact that events “010” and “100” occurred only in shorter patterns, and the events: “000”, “002”, “110”, “111”, “001”, and “112” persisted in longer frequent patterns.

Another aspect we observed from Table 4.1.2 is that both “002” and “000” have a higher probability to return to themselves, i.e. they are periodic. In addition, most of the frequent events have a high probability of moving to either “002” or “000”, and since these two events are periodic, we can infer that frequent patterns are periodic too. Furthermore, it explains why the most repetitive patterns are included in the following two patterns: "002-002-002-002-002-002-002-002-000", or "000-000-000-000-000-000-000-000-000-000-000-000-000-000-000-000-000".

One major limitation we faced throughout this task consists in the complexity of the data. Some sequences are too big; they can include even more than 500 events. This was a limitation to the SPAM algorithm because when sequences are too long a large number of candidates are generated, and it slows down the algorithm performance or in the worse case it results in running out of memory, especially when the minimum support threshold is small.

4.3.3 Marked point process model

One challenge we met in the simulation process is related to the computation of the covariance matrix used in the multivariate normal distribution. The problem arises when the observation time is long or when the value of the parameter λ is large. The time scale of the observed data is in seconds, and each user was observed at least for one hour. Therefore, using 3600 seconds as the observation time in the Poisson point process, a huge number of packet's arrival time points are generated resulting in large vectors of simulated upload quantity y_u and download quantity y_d. (each of y_u and y_d can contain more than 3000 simulated points). With such a large vector the computation of the correlation matrix takes a long time to execute and in the worse cases R runs out of memory.

The problem of running out of memory was solved by storing the covariance matrix as a band diagonal sparse matrix. In order to reduce the computation time, we converted the time scale into minutes and set the observation time to 60 minutes. However, another limitation associated with the structure of the observed burst data arose. This issue is due to the fact that a burst can aggregate a huge number of packets even when it lasted only a few seconds. For example, a burst that lasts less than a second can aggregate up to as many as 304 upload (or download) packets. In order to reach this level of aggregation we need a big value for the parameter λ, which takes us back to the initial problem.

We disregard this limitation and set the observation time to 60 minutes, and restricted the prior for λ from being higher than 100 (since beyond this value the simulation takes a long time, i.e simulating 1000 users can take more than three hours). The results, obtained by applying the standard rejection ABC algorithm, show that the model simulates lower values of upload and download sizes.

One reason of this flaw is that the value of the parameter λ is small. The mode of the posterior distribution of the parameter λ is 50.4868; with this value the maximum number of upload packets, for example, within a burst in the simulated data is 29, while in the observed data the maximum number of upload packets is 82003. Another reason of the model failure is the fact that λ is constant. Since λ is constant, the simulated inter-arrival times decay uniformly, and the larger λ is, the smaller becomes the inter-arrival time between two packets. This is the reason why a big value of λ is needed in order to be able to aggregate enough packets in one burst.

Due to the limitation described above, we could only run the standard rejection method since it is faster compared to ABC-MCMC algorithm. The ABC- MCMC algorithm would have taken too much time (we run it until 3 days).

4.3.4 Future research

The burst data aggregate many packets implying that various applications are included in one burst. It would be interesting to explore frequent patterns in burst sequences for single applications. This can help to understand and compare the effect of applications on the frequent patterns. Another extension is to divide the intervals defined on the upload and download variables, in the event coding, into finitely many intervals. Since high levels are not frequent at all, one suggestion would be to only divide the categories 0 and 1 defined on the upload and down-

load volume into small-grained categories . The time gap categories can remain unchanged because it has been noted that the time gap does not correlate with the frequent patterns.

Furthermore, n-gram models can be improved by combining the Good-Turing trigram model with lower-order n-gram models in order to produce accurate and less sparse estimates. One possible way is to express the trigram model as a mixture model of weighted unigram, bigram and trigram models, which is known as the linear interpolation[13]. Another way would be using the back-off methods [4]. If a particular event is not observed when a higher-order n-gram model is considered, back-off models allow the model to move backwards in the n-grams history and estimate the probability of the unseen event by a lower-order n-gram model that suffer less from the sparseness of the data.

Finally, in order to improve the Marked point process generative model, we suggest to simulate the arrival times of packets using a inhomogeneous Poisson point process, which allows to express λ as a function of time. Non constant λ enables to have irregular inter-arrival time intervals, which might be more suitable for the burst aggregation. Furthermore, we would recommend to interested researchers to implement the simulation model in a lower level language such as C or C++ which are faster than R.

5 Conclusions

The main task of this thesis was to propose and evaluate several methods for mobile traffic analysis based on IP data. This includes mining frequent patterns, formulating different n-gram models, and formulating a probabilistic model that simulates the IP burst data. The first two tasks were performed successfully, but we encountered computational difficulties in implementing our proposed probabilitic model. The model shows potential, however, and we feel confident that further research will overcome the encountered problems.

Applying SPAM algorithm on the IP burst database of 1819 sequences, with 40% as the minimum support threshold, the results show that only 8 out of 27 events are frequent and the most repetitive events are "000" and "002", in which the applications: " chat" and " other" are the most prevalent. More importantly, the results reveal that the frequent events are associated to the lower and middle level categories of upload and download packet sizes. Longer patterns include mostly the events in which both the upload and download packet sizes are in the same category, while shorter patterns include the events corresponding to upload and download sizes being in different categories.

Based on the perplexity evidence, the comparison of the formulated n-gram models (which include models based on Good-Turing, Laplace, and Jeffrey-Perks smoothing methods) suggest that the Good-Turing method provides the best results. Furthermore, it was revealed that most of the frequent events found by SPAM are periodic, and they have a higher probability to move into the events "000" and "002", which explains why the most frequent patterns are formed only by these two events.

Finally, this thesis provided a probabilistic model that simulates the burst log data. However, the model was not fully successful since it captured only behaviors of small bursts.

Bibliography

[1] Agrawal, R., Srikant, R., et al. (1994). Fast algorithms for mining association rules. In *Proc. 20th int. conf. very large data bases, VLDB*, volume 1215, pages 487–499.

[2] Ayres, J., Flannick, J., Gehrke, J., and Yiu, T. (2002). Sequential pattern mining using a bitmap representation. In *Proceedings of the eighth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 429–435. ACM.

[3] Beaumont, M. A., Zhang, W., and Balding, D. J. (2002). Approximate bayesian computation in population genetics. *Genetics*, 162(4):2025–2035.

[4] Chen, S. F. and Goodman, J. (1999). An empirical study of smoothing techniques for language modeling. *Computer Speech & Language*, 13(4):359–393.

[5] Cleveland, W. S. and Sun, D. X. (2000). Internet traffic data. *Journal of the American Statistical Association*, 95(451):979–985.

[6] Diggle, P., Heagerty, P., Liang, K.-Y., and Zeger, S. (2002). *Analysis of longitudinal data*. Oxford University Press.

[7] Fournier-Viger, P., Nkambou, R., and Nguifo, E. M. (2008). A knowledge discovery framework for learning task models from user interactions in intelligent tutoring systems. In *MICAI 2008: Advances in Artificial Intelligence*, pages 765–778. Springer.

[8] Gentle, J. E., Härdle, W., and Mori, Y. (2004). *Handbook of computational statistics*. Springer.

[9] Han, J., Kamber, M., and Pei, J. (2006). *Data mining: concepts and techniques*. Morgan kaufmann.

[10] Jabot, F., Faure, T., and Dumoulin, N. (2013). Easyabc: performing efficient approximate bayesian computation sampling schemes using r. *Methods in Ecology and Evolution*, 4(7):684–687.

[11] Jurafsky, D. and James, H. (2000). Speech and language processing an introduction to natural language processing, computational linguistics, and speech.

[12] Last, G. and Brandt, A. (1995). *Marked Point Processes on the real line: the dynamical approach*. Springer.

[13] Manning, C. D. and Schütze, H. (1999). *Foundations of statistical natural language processing*. MIT press.

[14] Marin, J.-M., Pudlo, P., Robert, C. P., and Ryder, R. J. (2012). Approximate bayesian computational methods. *Statistics and Computing*, 22(6):1167–1180.

[15] Pritchard, J. K., Seielstad, M. T., Perez-Lezaun, A., and Feldman, M. W. (1999). Population growth of human y chromosomes: a study of y chromosome microsatellites. *Molecular Biology and Evolution*, 16(12):1791–1798.

LIU-IDA/STAT-A--14/006—SE