SAPIENZA
UNIVERSITÀ DI ROMA

Doctoral Thesis

Network Communication Privacy: Traffic Masking against Traffic Analysis

A thesis submitted in partial fulfilment of the requirements for the degree of Doctor of Philosophy in

Information and Communication Technology

Author:
Alfonso Iacovazzi

Supervisor:
Prof. Andrea Baiocchi

Department of Information Engineering, Electronics and Telecommunications - DIET

March 2013

SAPIENZA - UNIVERSITY OF ROME

Abstract

Doctor of Philosophy

Network Communication Privacy:
Traffic Masking against Traffic Analysis

by Alfonso Iacovazzi

An increasing number of recent experimental works have been demonstrating the supposedly secure channels in the Internet are prone to privacy breaking under many respects, due to traffic features leaking information on the user activity and traffic content. As a matter of example, traffic flow classification at application level, web page identification, language/phrase detection in VoIP communications have all been successfully demonstrated against encrypted channels. In this thesis I aim at understanding if and how complex it is to obfuscate the information leaked by traffic features, namely packet lengths, direction, times. I define a security model that points out what the ideal target of masking is, and then define the optimized and practically implementable masking algorithms, yielding a trade-off between privacy and overhead/complexity of the masking algorithm. Numerical results are based on measured Internet traffic traces. Major findings are that: i) optimized full masking achieves similar overhead values with padding only and in case fragmentation is allowed; ii) if practical realizability is accounted for, optimized statistical masking algorithms attain only moderately better overhead than simple fixed pattern masking algorithms, while still leaking correlation information that can be exploited by the adversary.

Contents

List of Figures

List of Tables

Chapter 1

Introduction: From Traffic Analysis to Traffic Masking

A number of works over the last few years have given extensive experimental evidence that even within a secure channel, a packetized flow leaks information to an adversary through observation of features of traffic flows, e.g., the ordered sequence of packet lengths, packet inter-arrival times, packet directions. Based on these information that is available to an adversary even when the flow is carried within a secure channel (e.g. SSL/TLS or SSH connections), it has been shown that a number of approaches yields feasible algorithms to identify the type of service or application protocol run among a given set of alternatives (*traffic classification*). More generally, *traffic analysis* can identify different types of user activities within a secure channel, e.g. discriminate between web navigation versus remote management or file transfer, or even the type of search carried out by a user of Google within a "secure" web session. Other privacy breaking attacks based on statistical analysis of packet flow features have been demonstrated, e.g. to profile web access, to infer language of phone calls.

This communication privacy break is a positive proof that ciphering does not conceal all relevant information of a packetized, discontinuous application flow; hence in this Thesis I aim at describing the investigation of privacy *protection* against traffic analysis I have done during my PhD. Besides being a privacy issue, traffic analysis tools can be useful to network administrators and operators for enforcement of security policies and traffic filtering, or to support quality of service mechanisms. A key point for the robustness of those "legitimate" uses of traffic analysis tools is to check how much effort is needed to fool them. Those reasons motivate us to investigate how traffic feature leakage can be concealed to "adversaries" exploiting traffic analysis. We term this *traffic masking*.

Packet length and direction masking can exploit two basic mechanisms: padding and fragmenting. Padding amounts to the insertion of a number of extra bit in the packet, that can be stripped off by the recipient so that information is not corrupted, but such that the adversary measures an altered value of packet length of the ciphered packet. Padding can come in two forms: adding bits to a packet provided by a host or creating a fake packet (*dummy* packet, that will be discarded altogether at destination). Fragmenting is another form of packet length modification: from a single original packet with payload L, n packets spring out, with payload lengths L_j, $j = 1, \ldots, n$, such that $L = L_1 + \cdots + L_n$. Overhead must be added to the newly generated packets to allow correct reassembly of the original packet at destination[1]. Masking of packet inter-arrival times is essentially based on insertion of dummy packets and delaying of packets. There is a price to be paid for masking via padding (including dummy packets), fragmentation and delaying: bytes or even entire

[1] As a matter of example, fragmenting IP packets entails an extra header overhead of 20 bytes for each fragment, after the first one.

new packets are added to the original traffic flow (*byte overhead*), and a delay is imposed to the original packet flow (*time overhead*).

As for contributions of this Thesis, after analyzing the research field of the statistical classification for Internet traffic, we formally define the problem of privacy against traffic classification, thus finding what the ideal traffic masking should do. Then, we define the achievable performance bounds for a masking algorithm, by defining an optimization problem to find an ideal masking algorithm that minimizes overhead cost. The study for finding masking techniques for perfect privacy was first applied only to the packet lengths in a flow and later extended to all features of the traffic. Following, in order to further reduce overhead introduced by the algorithms proposed, we have relaxed the hypothesis of ideal masking to obtain partial masking allowing the control of information leakage carried by statistical features of the traffic. At the end, we tackle the problem of masking packet traffic flow carried in an encrypted channel, irrespective of the application(s) it comes from. After the theoretical studies, we report in each Section the results obtained with the masking algorithms developed.

In the rest of this Chapter, we describe what is the state of the art in the field of traffic analysis, what are the reasons behind the development of masking techniques, and then we move to describe the state of the art in privacy preserving against traffic analysis.

1.1 Traffic Analysis: State of the Art

One of the biggest concern in the Internet world is to provide *perfect security* to the users, with particular attention to user privacy. The typical answer to these privacy concerns is to simply encrypt the data going through the network

in order to conceal the information from a possible attacker intercepting traffic. However, this alone is not enough since several features of the encrypted network data, such as packet sizes, timing, direction of packets, the number of objects downloaded from a web page, the number of components of these objects, and the number of connections, can still leak information about the traffic.

Such *side-channel information leaks* have been widely studied for a decade, in the context of secure shell (SSH) [1], video-streaming [2], voice-over-IP (VoIP) [3], web browsing and others. Particularly, a line of research conducted by various research groups has studied anonymity issues in encrypted web traffic. For example, it has been shown that since each web page has a distinct size, and usually loads some resource objects (e.g., images) of different sizes, the attacker can fingerprint the page so that even when a user visits it through HTTPS, the page can be re-identified [4].

Classification is the problem of identifying to which of a set of categories a new observation belongs, on the basis of a training set of data containing observations (or instances) whose category membership is known. The individual observations are analyzed into a set of quantifiable properties, known as *features*.

Traffic analysis and classification have been widely studied and employed in literature; in [5] we can find an exhaustive survey on the traffic analysis problem and on the most important protocols, attacks and design issues. Among all the attacks presented, timing attacks, communication pattern attacks, packet counting attacks, and intersection attacks are the most used traffic analysis attacks.

1.1.1 Application Protocol Identification

As broadband communications widen the range of popular applications, there is an increasing demand of fast traffic classification means according to the services that generate data flows [6]. The specific meaning of service depends on the context and purpose of traffic classification. In case of traffic filtering for security or policy enforcement purposes, service can be usually identified with application layer protocol. Another example of usage of application traffic classification is QoS related to differentiate traffic management according to services carried by different traffic flows. A sufficiently robust classifier could be a useful element in implementing differentiated QoS without deploying complex traffic engineering schemes that require cooperation with end hosts.

Currently available techniques for traffic classification are: port based analysis, deep packet inspection and statistical based system. Port analysis consists of examining the port number of TCP/UDP headers and mapping them to application as defined by ICANN (formerly IANA). This method is becoming ineffective because of applications running on non-standard ports (e.g. peer-to-peer). For this matter, traffic classification at application level is based on the analysis of the entire packet content, header plus payload, by looking for specific application protocol signature. This is so called, deep packet inspection (DPI). There are widely available tools for such a classification approach (e.g. L7filter, BRO, Snort [7]). They can be very accurate, but when catching up with high speed links, i.e. for backbone use (Gbps links), they result too expensive in terms of computational power and storage resource. Moreover DPI fails in classifying application carried by encrypted flows (i.e. Secure VPN). This point and the increasing fraction of traffic carried by ISPs encrypted within VPN tunnels, makes actual classification system ineffective. In this scenario statistical approach can be a direction for effective traffic classification (see [6]).

Different approaches to traffic classification have been developed, using information available at IP layer such as packet inter-arrival times, packet sizes, and overall amount of bytes transferred. Callado et al.[6], present a good tutorial to approach the problem of traffic classification and point out open research issues, as well as a comparison in terms of accuracy and completeness of two identification methods. As for specific traffic classification approaches, some proposals [8, 9] need complete TCP flows as input (off-line classification).

In [10] Karagiannis et al. developed a heuristic that uses social, functional and application level behaviors of a host to identify all traffic flows originating from it. This approach, although really innovative, is tailored onto a specific source host.

Crotti et al. [11] use only size and inter-arrival time of first m packets to create a statistical descriptor (a fingerprint) of an application layer protocol. This fingerprint is then used to measure the similarity of a given flow to the corresponding protocol.

Moore et al. [9] use a supervised machine learning algorithm, the well known Naïve Bayes (and its generalization, Kernel Estimation) on a wide set of characteristics (tens or hundreds), as flow duration, packets inter-arrival time and payload size and their statistics (mean, variance...). Moreover, they use a filtering technique to identify the best characteristics to be used with the mentioned methods.

A number of works [8, 12, 13] rely on unsupervised machine learning techniques as K-means is. McGregor et al. [8] explore the possibility to use cluster analysis to group flows using transport layer attributes, but they do not evaluate the accuracy of the classification. Zander et al. [12] extend this work by using another Expectation Maximization (EM) algorithm named Autoclass. They also analyze the best set of attributes to use. Both these works only test Bayesian clustering technique trained by an EM algorithm, which has a

slow learning time. Bernaille et al. [13] use faster clustering algorithms representing data in different spaces: K-means and Gaussian Mixture Models (GMM) for Euclidean space and Spectral clustering in Hidden Markov Model (HMM) based space. The only features they use are packet size and packet direction: they demonstrate the effectiveness of these algorithms even using a small number of packets (e.g. the first four of a TCP connection).

Hidden Markov Model theory is also used in [14]. Packets size and inter-arrival time are used to build a model describing a given protocol. Even though this approach can classify encrypted applications, its performance with SSH flows is 76% detection rate and 8% false negative, which is not as good as with other common application flows such as WWW and instant messaging.

Alshammari et al [15] attempt to classify/identify applications services running on SSH by exploiting two supervised learning algorithms such as AdaBoost and RIPPER by relying on IP packet lengths, duration of the flows and arrival times. Results indicate detection rate up to 99% and 0.7% of false positive by exploiting RIPPER classifier, but the classifiers work only off-line, i.e. the entire flow must be available.

Concerning SSH encrypted applications, Dusi et al. [16] approached classification by exploiting GMM and SVM (Support Vector Machine) based techniques. They achieved accuracy up to 99.2% by analyzing four encrypted packets after SSH handshake. They collected artificial traffic traces by developing a tool based on SSH, which forwards four applications (HTTP, POP3, POP3S and Emule), however the testbed is quite artificial and far from the actual use of the protocol in real networks.

The flow-based classification mechanisms proposed so far cannot deal with network-layer tunneling techniques, such as the ones provided by IPsec. The flows are multiplexed into the same encrypted connection and there is not a reliable way to reassembly the flows routed through the IPSec channel by

observing only the encrypted traffic.

1.1.2 Webpage Fingerprinting

Beyond the classification of application classes there are other privacy leaking that have been studied in the literature. For example there are a lot of work showing how recognize webpages downloaded by a user. In this context an attacker can fingerprint web pages by their side-channel characteristics, then eavesdrop on the victim user's encrypted traffic to identify which web pages the user visits.

An attack demo was described in a course project report in 1998 by Cheng et al [17]. Sun et al [4] and Danezis [18] both indicated that this type of side-channel attack defeats the goal of anonymity channels, such as Tor, Mix-Master and WebMixes. Sun et al's experiment showed that $100,000$ web pages from a wide range of different sites could be effectively fingerprinted. Besides SSL/TLS, Bissias et al conducted a similar experiment on WPA and IPSec [19].

Even Hintz in [20] analysed the concept of website fingerprint. When a user visit a typical webpage, he downloads several files: the HTML file, the images included in the page, and referenced stylesheets. For example if a user visited CNN's webpage at `www.cnn.com`, he would download 40 separate objects each with a certain size. The set of transfer sizes for a given webpage comprises that page's fingerprint. Webpages with a large number of objects have fingerprints composed of many different sizes. The more files in a fingerprint, the larger the chance that the fingerprint will be unique. The Author also states that an eavesdropper observing the traffic can infer the webpages visited by the client by observing those fingerprints threatening so users' privacy.

Liberatore et al. in [21] evaluate traffic analysis techniques that infer the source of a webpage retrieved under the cover of an encrypted tunnel by

comparing observed traffic to profiles (fingerprints) of known sites created from packet lengths, and are referred to as profiling attacks (fingerprinting attacks).

1.1.3 Information Leaks in VoIP Traffic

It is widely accepted that, due to the sensibility of private conversations, VoIP traffic should be encrypted before transmission over the Internet. The current focus on VoIP security has centered around efficient techniques for ensuring confidentiality of VoIP conversations, but no efforts have been done to evaluate how traffic analysis could threat the privacy of a conversation.

In their article, Wright et al. [22] exploit the fact that using bandwidth-saving techniques, such as variable bit rate (VBR) coding, implies that the size of a VoIP packet is directly determined by the type of sound its payload encodes. In VBR mode, the encoder takes advantage of the fact that some sounds are easier to represent than others. For example, with Speex, the coder Authors decided to use, vowels and high-energy transients require higher bit rates than fricative sounds like "s" or "f". To achieve improved sound quality and a low (average) bit rate, the encoder uses fewer bits to encode frames which contain "easy" sounds and more bits for frames with sounds that are harder to encode. Because the VBR encoder selects the best bit rate for each frame, the size of a packet can be used as a predictor of the bit rate used to encode the corresponding frame. Therefore, given only packet lengths, it is possible to extract information about the underlying speech.

Again Wright in [3] show that an eavesdropper who has access to neither recordings of the speaker's voice nor even a single utterance of the target phrase, can identify instances of the phrase with average accuracy of 50%.

In [23], White et al. showed how it is possible to derive approximate transcripts of encrypted VoIP conversations by segmenting an observed packet

stream into distinct subsequences representing individual phonemes and classifying those subsequences by the phonemes they encode.

The approach they pursued leverages the correlation between voiced sounds and the size of encrypted packets. Specifically, they showed how it is possible to segment a sequence of packet sizes into subsequences corresponding to individual phonemes and then classify these subsequences by the specific phonemes they represent. Then authors proved the possibility of segmenting such a phonetic transcript on word boundaries to recover subsequences of phonemes corresponding to individual words and map those subsequences to words, thereby providing a hypothesized transcript of the conversation.

In their approach, first a *maximum entropy* model is used to segment the sequence of packet sizes into subsequences corresponding to individual phonemes. Then a combination of *profile hidden Markov* models and maximum entropy is applied to classify each subsequence of packet sizes according to the phoneme the subsequence represents, resulting in an approximate phonetic transcript of the spoken audio. The hypothesized transcript is improved by applying a trigram language model over phonemes (and phoneme types) which captures contextual information, such as likely phoneme subsequences, and corrects the transcript to represent the most likely sequence of phonemes given both the classification results and the language model. Next, the resulting transcript is segmented into subsequences of phonemes corresponding to individual words using a phonetic constraint model. Finally, each subsequence is matched to the appropriate English word using a phonetic edit distance metric.

1.2 Traffic Masking: State of The Art

As outlined in the previous Paragraph, a number of works over the last years have given extensive experimental evidence that an encrypted packetized flow leaks information about the user activity to an observer through the ordered sequence of packet lengths, packet inter-arrival times, packet directions and other detectable features. Due to the features analyzed, this communication privacy break is a positive proof that ciphering does not conceal all relevant information of a packetized traffic flow.

By quantizing these features (e.g., padding packets to fixed sizes), the amount of information that is leaked can be minimized, but at the cost of degrading the efficiency and performance of the underlying network protocols. Certainly, one can pad all encrypted packets such that their sizes are always equal to that of the maximum transmission unit (MTU), but for many network protocols doing so would more than double the amount of data sent. For these protocols, such excessive padding is simply not a satisfactory solution to the problem.

Moreover, the performance of encrypted network protocols often takes precedence over privacy concerns in practical applications. While it may be possible to allow users to tune the tradeoff between efficiency and privacy to their liking, there is often no clear meaning in terms of the levels of privacy and performance associated with such actions. As a consequence of this bias towards efficiency, several security-oriented network protocols have been found to leak more information about the underlying data than originally thought.

1.2.1 Existing Masking Architectures

Some countermeasures against traffic analysis are already supported by the main architecture of communications encryption, such as SSH, TLS and IPSec.

The countermeasures used have the main purpose of concealing the loss of information determined by the lengths of the packets exchanged.

SSH: RFC 4253 [24] specifies that padding can be of arbitrary length, such that the total length of packet is a multiple of the cipher block size or 8, whichever is larger. There MUST be at least four bytes of padding. The padding SHOULD consist of random bytes. The maximum amount of padding is 255 bytes. In the basic OpenSSH implementation, the padding length will depend on the payload and the cipher block size. So although the padding itself is random, the final packet size will be just a step function of the payload size.

SSL: Also the specifications of TLS protocol (versions 1.1 and 1.2) described in RFCs 4346 and 5246, offer the possibility to add some padding in order to alterate the packet lengths. Even here, padding that is added to force the length of the plaintext must be an integral multiple of the block cipher's block length. The padding may be any length up to 255 bytes. The choice of possible techniques for adding padding is left to the discretion of the individual implementations. GnuTLS is one of the most famous secure communications library implementing the SSL, TLS protocols. If properly enabled, it allows to add a random padding with uniform distribution.

IPSec: IPSec is a standards of network level that provides various cryptographic algorithms in order to provide security services. Even IPSec allows to add a random padding with uniform distribution.

In 2007, Kiraly et al. [25, 26] have proposed a new framework based on IPSec, which allows to integrate techniques of Traffic Flow Confidentiality (TFC). The authors have designed a new architecture that can be considered as a specific substrate maintaining backward compatibility with traditional IPsec implementations. The developed architecture is structured into two main modules: one deals with the logical control of TFC procedures and algorithms,

while the other one performs all the functionalities for alterating the features of the input packets. This latter module is, in turn, divided into several components, each of which allows to:

1. queue packets before sending,
2. extract the packets from the queue according to a timing procedure,
3. manage packet lenghts with the possibility of introducing padding,
4. generate dummy packets.

The framework has been developed into the Linux kernel, fully integrated with IPSec in order to take advantage of all features offered by it.

An interesting architecture to support communications like VoIP and Instant Messaging such that it can be robust against traffic analysis, was conducted by Danezis et al. and is described in detail in [27]. The basic idea of this framework is to create a social network consisting of a number of nodes, each of them is in contact with a set of *friends* through a connection considered unobservable by an adversary, whilst all other connections, called *bridges*, are observable. This structural hypothesis of the network can offer unobservability, i.e. an opponent will not be able to understand who is talking to whom.

In order to fully ensure unobservability, the framework provides the opportunity to integrate padding and dummying techniques such as full padidng or those described in [28]. No delaying support is provided to prevent timing attacks.

Other works have been developed in recent years [29], to ensure unobservable communications by an unauthorized third party and under certain

assumptions. The architectures proposed aim mainly at not allowing identification of the source and/or receiver nodes for a given flow. For this reason problems relating to traffic analysis are often not thorough.

1.2.2 Trivial Algorithms

There are some easy countermeasure with padding mechanisms that are not easily supported in existing encrypted network protocol standards due to the amount of padding added [30]. In this scenario, we assume the countermeasure will be capable of managing fragmentation and padding of the data before calling the encryption scheme.

- *Linear padding:* All packet lengths are increased to the nearest multiple of 128, or the MTU, whichever is smaller.
- *Exponential padding:* All packet lengths are increased to the nearest power of two, or the MTU, whichever is smaller.
- *Mice-Elephants padding:* If the packet length is $T \leq 128$, then the packet is increased to 128 bytes; otherwise it is padded to the MTU.
- *Pad to MTU:* All packet lengths are increased to the MTU.
- *Packet Random MTU padding:* Let M be the MTU and ℓ be the input packet length. For each packet, a value $r \in \{0, 8, 16, \ldots, M - \ell\}$ is sampled uniformly at random andthe packet length is increased by r.

1.2.3 Recent Algorithms Proposed

During the years various countermeasures have been developed. Some of them are feature distribution-based as the work of Wright et al. [28]. They presented two novel suggestions as improvements upon traditional per-packet padding

countermeasures: direct target sampling (DTS) and traffic morphing (TM). On the surface, both techniques have the same objective. That is, they augment a protocol's packets by chopping and padding such that the augmented packets appear to come from a pre-defined target distribution (i.e., a different web page). Ideally, DTS and TM have security benefits over traditional per-packet padding strategies because they do not preserve the underlying protocol's number of packets transmitted nor packet lengths.

Direct target sampling: Given a pair of web pages A and B, where A is the source and B is the target, we can derive a probability distribution over their respective packet lengths, D_A and D_B. When a packet of length L_i is produced for web page A, we sample from the packet length distribution D_B to get a new length L'_i. If $L'_i > L_i$, we pad the packet from A to length L_i and send the padded packet. Otherwise, we send L'_i bytes of the original packet and continue sampling from D_B until all bytes of the original packet have been sent. Wright et al. left unspecified morphing with respect to packet timings.

Traffic morphing: Traffic morphing operates similarly to direct target sampling except that instead of sampling from the target distribution directly, we use convex optimization methods to produce a morphing matrix that ensures we make the source distribution look like the target while simultaneously minimizing overhead. Each column in the matrix is associated with one of the packet lengths in the source distribution, and that column defines the target distribution to sample from when that source packet length is encountered. As an example, if we receive a source packet of length L_i, we find the associated column in the matrix and sample from its distribution to find an output length L'_i. One matrix is made for all ordered pairs of source and target web pages (A, B). The process of padding and splitting packets occurs exactly as in the direct target sampling case. Like the direct target sampling method, once the source web page stops sending packets, dummy packets are sampled directly

from D_B until the L1 distance between the distribution of sent packet lengths and D_B is less than 0.3.

Again with the idea to camouflage the web page visited by a user, Luo et al. [31] propose a new strategy to obfuscates the encrypted traffic by exploiting the protocol features in TCP and HTTP. They developed a new framework able to modify statistical features of an original flow by using a set of traffic transformations both at the application and at the transportation layer. For a given flow, it can be applied a transformation as combination of these techniques: packet padding and/or fragmentation, HTTP Range, MSS negotiation, advertising window, message retransmissions, HTTP pipelining, packet delay.

Shui Yu et al.[32–34] implemented a new strategy of packet padding aiming at offering perfect anonymity on web browsing. Their proposal comes from the fact that users generally access a number of web pages at one web site according to their own habits or interests. This has been confirmed by applications of web caching and web page prefetching technologies. The proposed solution allows to disguise the fingerprints of web sites at the server side by injecting predicted web pages that users are going to download as cover traffic, rather than using dummy packets as cover traffic. So Authors conclude that from a long term viewpoint, this novel strategy wastes limited bandwidth and causes limited delay.

Instead, Zhang at al. [35] contrast traffic analysis by means of traffic reshaping technique. By exploiting multiple virtual MAC interfaces, an application flow is dynamically subdivided in a set of new flows and then dispatched among these interfaces, and different traffic features are reshaped on each virtual interface to hide those of the original traffic.

1.2.4 Open Problems

Although a significant amount of previous work has investigated the topic of Traffic Analysis countermeasures, and specifically the case of preventing website identification attacks, the results were largely incomparable due to differing experimental methodology and datasets.

Furthermore, in [30] Dyer et al. show that it is still possible to classify traffic flows after masking. They consider nine masking countermeasures applied to web pages, and adopt some machine learning algorithms (Naïve bayes, multinomial Naïve bayes and support vector machine) to identify which of two web pages was downloaded. Accuracy of 98% is obtained, and they conclude that more investigation is necessary to effectively conceal the whole information leakage.

Chapter 2

Classification Problem

A sequence of works detail a variety of *Traffic Analysis* attacks, in the form of classifiers that attempt to identify the application generate a flow over an encrypted channel (see Chapter 1). These classifiers use supervised machine learning algorithms, meaning they are able to train on traces that are labeled with the destination website. Each algorithm has a training and a testing phase. During training, the algorithm is given a set $\{(X_1, w_1), (X_2, w_2), \ldots, (X_n, w_n)\}$, where each X_i is an vector of features and w_i is a label. During testing the classification algorithm is given a vector Y and must return a label. In our case, a vector X_i contains information about the lengths, timings, and direction of packets in the encrypted connection containing an application flow, and the format of a vector X_i is dependent upon the classifier. In the remainder of this Chapter, we describe the model of the traffic flow classifier, define which are the features usable for classification and give the maximum success probability of the ideal flow classifier. At the end we briefly present the machine learning used for the performance evaluation in this Thesis and describe the dataset creation process.

2.1 Model of a Traffic Flow Classifier

We consider a packet network where we identify two end points (hosts) running a given application. The information flow between the two hosts is ciphered and authenticated (message authentication). In spite of using a secure channel for communication, still there is information leaking to an adversary observing the information flow between the two hosts.

Any application traffic flow between an initiator entity A and the responder entity B (e.g., client and server for the given flow, respectively) can be cast into a sequence of $N \geq 1$ message bursts[1]. Each burst consists of one or more messages in the $A \rightarrow B$ direction or in the opposite direction $B \rightarrow A$. Bursts in the two opposite directions alternate, starting from the initial burst sent by the initiator A to the responder B. Figure 2.1 depicts an example of two flows, where A and B sides are represented by vertical lines and time increases downward.

A full description of the flow is obtained with:

- the vector $\mathbf{K} = [K_1, \ldots, K_N]$ of the number of messages in each burst;
- message lengths in each burst, denoted as $\mathbf{L}_i = [L_i(1), \ldots, L_i(K_i)]$, for $i = 1, \ldots, N$;
- message epochs[2], denoted as $\mathbf{T}_i = [T_i(1), \ldots, T_i(K_i)]$, for $i = 1, \ldots, N$; the time origin is set as the time epoch associated to the first packet of the entire flow, i.e., $T_1(1) = 0$; therefore, $T_i(r)$ represent the time elapsed since the beginning of the flow up to the r-th message of the i-th burst (relative timing). We will use also *message gap times*, defined as $\Delta t_i(r) = T_i(r+1) - T_i(r)$, for $r = 1, \ldots, K_i$, with $T_i(K_i + 1) \equiv T_{i+1}(1)$.

[1] We use the generic term "message", since the model is applicable to general traffic flows, from web pages to IP layer flows.

[2] In practice, time epochs refers to time-stamps associated to packets collected by the adversary at the traffic flow capture point within the network.

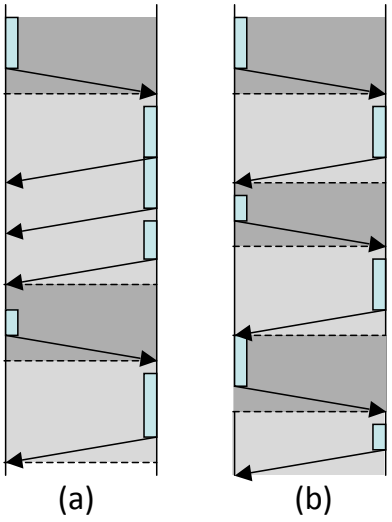

FIGURE 2.1: Examples of message exchanges of two application flows: (a) one way data transfer, like http; (b) alternate message sending, like most signaling and control protocols.

In the example flows shown in Figure 2.1 the flow labeled (a) has $N = 4$, with $\mathbf{K} = [1, 3, 1, 1]$. For flow (b) it is $N = 6$ and $\mathbf{K} = [1, 1, 1, 1, 1, 1]$.

A more compact description is obtained by considering only *burst sizes* $B_i = \sum_{r=1}^{K_i} L_i(r)$ and *burst epochs* $\Theta_i = T_i(1)$, for $i = 1, \ldots, N$; hence a feature vector $\mathbf{X} = [\mathbf{B}, \mathbf{\Theta}]$.

2.2 A General Model of an Application Flow

Let the feature vector associated to a flow be $\mathbf{X}$. According to our definition, a full description of the flow entails $\mathbf{X} = [\mathbf{K}, \mathbf{L}, \mathbf{T}]$. In general, the feature vector can be a subset of the full description, according to adversary aims and resources, e.g., for real time classification only the first m messages of a flow could be considered, with m typically ranging from one up to several units.

In case of web pages classification, more aggregate features could possibly be used by the adversary, e.g., the amount of bytes sent per burst or even the overall amount of bytes sent with the flow in either direction.

Each entry of $\mathbf{X}$ is modeled as a positive, discrete random variable[3] with a finite support $[1, \ldots, \ell]$ for some suitable constant ℓ. Let Ω denote the state space of $\mathbf{X}$.

We define a privacy model, where each flow can belong to one of M classes. An adversary can observe flow features by means of network traffic analysis and aims at identifying the class each flow belongs to. The example we focus on is *flow classification*, where the classes are applications, but results of this Section hold in general for classification problems.

Let us consider M applications, denoted by a label $\mathcal{A} \in \{\mathcal{A}_1, \ldots, \mathcal{A}_M\}$. We let $p_j(\mathbf{x}) = \mathcal{P}(\mathbf{X} = \mathbf{x}|\mathcal{A} = \mathcal{A}_j)$, $\mathbf{x} \in \Omega, j = 1, \ldots, M$, be the *probability distribution function* (pdf) of the feature vector, conditional on the flow belonging to the j-th application; we further denote with $P_j = \mathcal{P}(\mathcal{A} = \mathcal{A}_j)$ the a priori probability that a flow belongs to application $\mathcal{A}_j$.

In general, the traffic masking operation includes introducing dummy flows, to modify the a priori probabilities P_j into new values Q_j, and transforming each flow sent through the network so that the output flow features are given by $\mathbf{Y} = \phi(\mathbf{X}; \mathcal{A})$, thus altering the original feature pdf. The flow transformation implies message padding, fragmenting, insertion of dummy messages, message delaying. This transformation and the relevant modification of feature values can depend in general on the application the input flow belongs to, which must be known at the masking device. This is made explicit by highlighting the parameter $\mathcal{A}$ in the mapping $\phi(\cdot; \mathcal{A})$. This mapping induces a

[3] While $\mathbf{K}$ and $\mathbf{L}$ are natively discrete, the time values $\mathbf{T}$ can be made discrete by specifying a time quantum, e.g., the time measurement resolution or a small multiple thereof.

probability measure on $\mathbf{Y}$, denoted by $q_j(\mathbf{y}) = \mathcal{P}(\mathbf{Y} = \mathbf{y}|\mathcal{A} = \mathcal{A}_j)$. We assume that the state space of $\mathbf{Y}$ is the same as that of $\mathbf{X}$, namely Ω.

As matter of example, if $\mathbf{X}$ reduces only to message lengths, a possible transformation is padding, i.e., $\mathbf{Y} = \mathbf{X} + \mathbf{U}$, $\mathbf{U}$ being a non negative vector, whose characteristics depend on the application $\mathbf{X}$ belongs to.

We assume an *eavesdropping* adversary, aiming at flow classification. The adversary can observe ciphered and masked flows (including dummy flows) and can detect the feature vector $\mathbf{y}$ for each observed flow. In other words, the adversary can collect samples $\mathbf{y}$ of the random variables $\mathbf{Y}$. Moreover, the adversary knows ciphering and masking algorithms used in the secure channel, and is given knowledge of the conditional pdfs of the masked flow features $q_j(\cdot)$[4]. The aim of the adversary is to guess the application the original flow belongs to. This is summarized by an algorithm named $TA(\mathbf{Y}) : \Omega \rightarrow \{1, \ldots, M\}$ yielding the application label for the observed masked flow feature vector.

An overall scheme of the masking plus enciphering at sending side is shown in Figure 2.2. The reverse operations (deciphering and de-masking) take place at receiver side, the latter by using overhead information, e.g., to identify dummy messages and padding. Dummy flows are added to modify the a priori pdf P_j into Q_j, for $j = 1, \ldots, M$. Ideally, the Q_j's should be uniform ($Q_j = 1/M$). Coupled with perfect masking this brings the adversary success probability of correctly classifying observed flows to its theoretical minimum $1/M$. If dummy flows are suppressed to save overhead, the a priori application pdf can be exploited by the adversary, if known. Clearly, if one class is overwhelmingly more probable the the others, simple guessing gives the adversary

[4] As a matter of example, the adversary has a database containing a set of masked flows for each application, with metadata assessing the application that originated those flows; such a database can be used to train a classification algorithm; this is similar to a known/chosen plaintext eavesdropper model, depending on the way the database is constructed.

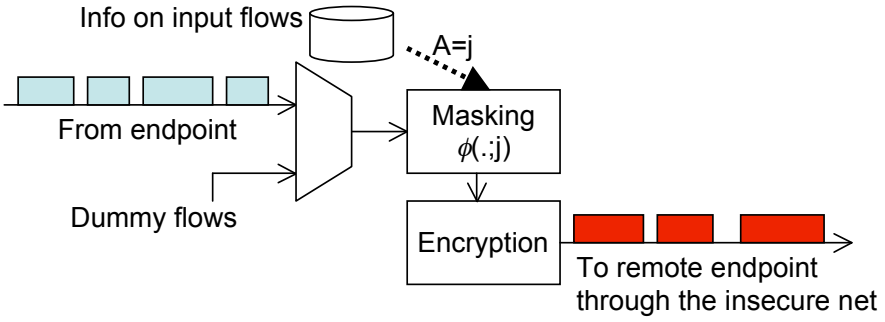

FIGURE 2.2: Traffic flow masking block scheme at sending side: dummy flows are added to modify the a priori pdf of generating applications.

a high success probability, i.e., the endpoint traffic is highly predictable.

Given the adversary model above, by *full masking* we mean removing any information leakage that could be exploited by the adversary to classify observed flows. With full masking, the success probability of the adversary is minimized to $1/M$. Conversely, *partial masking* lets some information leak by establishing a trade-off between privacy and feasibility/complexity/overhead-/delay of masking. We also define *ideal masking* an algorithm that makes use of the knowledge of the whole feature vector of a flow to mask every message of it, whereas *practical masking* refers to the subset of algorithms that can run in real time, by processing messages of the masked flow as they arrive, independently at the two endpoints. A practical masking algorithm can use features of the first k messages to decide on masking of the $(k+1)$-th one.

In the next Section we define the adversary classifier function $TA(\cdot)$ with maximum probability of success and derive conditions on the masking transformation $\phi(\cdot;\mathcal{A})$ for full masking.

2.3 Success Probability of the Ideal Flow Classifier

Application traffic flow classification can be cast into a hypothesis test problem, with simple hypotheses $\mathcal{A} = \mathcal{A}_j$. This is the classical Nyman-Pearson test, that is known to be the optimum (Uniformly Most Powerful) test for this kind of problem [36]. The state space Ω is partitioned into M decision regions $\mathcal{D}_i$, such that the adversary sets $TA(\mathbf{y}) = i$ iff $\mathbf{y} \in \mathcal{D}_i$, $i = 1, \ldots, M$. Then, the probability of success of the statistical test is

$$P_{succ} = \sum_{i=1}^{M} Q_i \sum_{\mathbf{y}\in\Omega} \mathcal{P}(TA(\mathbf{y}) = i|\mathcal{A} = \mathcal{A}_i) = \sum_{i=1}^{M} Q_i \sum_{\mathbf{y}\in\mathcal{D}_i} q_i(\mathbf{y}) \tag{2.1}$$

Let us assume that there exists $\bar{\mathbf{y}} \in \mathcal{D}_i$ such that $Q_i q_i(\bar{\mathbf{y}}) < Q_j q_j(\bar{\mathbf{y}})$ for some $j \neq i$. Then, we can replace the decision regions $\mathcal{D}_i$ with $\mathcal{D}'_i = \mathcal{D}_i \setminus \{\bar{\mathbf{y}}\}$, $\mathcal{D}'_j = \mathcal{D}_j \bigcup \{\bar{\mathbf{y}}\}$ and $\mathcal{D}'_k = \mathcal{D}_k$, $k \neq i, j$. The value of P_{succ} with these new decision regions is

$$P'_{succ} = \sum_{i=1}^{M} Q_i \sum_{\mathbf{y}\in\mathcal{D}'_i} q_i(\mathbf{y}) \geq P_{succ} \tag{2.2}$$

This shows that the decision algorithm TA that maximizes P_{succ} must define decision regions according to $\mathcal{D}_i \equiv \{\mathbf{y} : Q_i q_i(\mathbf{y}) \geq Q_j q_j(\mathbf{y}) \ \forall j \neq i\}$. With these regions, we have

$$P_{succ} = \sum_{\mathbf{y}\in\Omega} \max\{Q_1 q_1(\mathbf{y}), \ldots, Q_M q_M(\mathbf{y})\} \tag{2.3}$$

The success probability can be minimized by making the decision variable $\mathbf{Y} = \phi(\mathbf{X}, \mathcal{A})$ such that $q_1(\mathbf{y}) = \cdots = q_M(\mathbf{y})$. With this choice the success probability reduces to $\max\{Q_1, \ldots, Q_M\}$, that is the value obtained by simple biased guessing. If further dummy flows are added so as to obtain $Q_i = 1/M$, then $P_{succ} = 1/M$. This value is achievable with a trivial classification

algorithm that guesses $\mathcal{A}$ at random. The equalization of the probabilities Q_i can be obtained by adding dummy flows as follows. With no loss of generality, assume $P_1 \geq P_2 \geq \cdots \geq P_M$ and[5] $P_1 > 1/M$. For each flow sent out a dummy flow is produced with probability $1 - 1/(MP_1)$; the dummy flow is of class i (i.e., it has the same statistical properties of flows generated by application i) with probability $(P_1 - P_i)/(MP_1 - 1)$ for $i = 2, \ldots, M$.

Summing up, *perfect privacy* or full masking, as we name it here means that any leakage about application is removed off the masked flow features and entails finding a transformation of the original flow so that the output features have a same pdf irrespective of the input application, i.e., $Q_i = 1/M$ and $q_1(\mathbf{y}) = \cdots = q_M(\mathbf{y})$, $i = 1, \ldots, M$.

2.4 Machine Learning based Classifiers

In this Section the classification techniques that have been used in this work to evaluate information leakage before and after masking will be presented.

2.4.1 Naïve Bayes Classifier

A Naïve Bayes classifier is a simple probabilistic classifier based on applying Bayes' theorem with strong (Naïve) independence assumptions. A more descriptive term for the underlying probability model would be "independent feature model". In simple terms, a Naïve Bayes classifier assumes that the presence (or absence) of a particular feature of a class is unrelated to the presence (or absence) of any other feature, given the class variable.

Given a training set $x = (x_1, \ldots, x_n)$ such that each instance x_i is described by M features $\{f_1^i, f_2^i, \ldots, f_M^i\}$ that can take numeric or discrete values.

[5] If $P_1 = 1/M$, then all P_i would be equal to $1/M$ and there would be no need of dummy flows.

Assume now that there are K known classes of interest. Let $\mathbf{C} = \{c_1, \ldots, c_K\}$ represent the set of all known classes. For each observed instance x_i there a known mapping $C : \mathbb{R}^M \rightarrow \mathbf{C}$ representing the membership of instance x_i to a particular class of interest.

Bayesian statistical conclusions about the class c_k of an unobserved flow $y = \{f_1, f_2, \ldots, f_M\}$ are based on probability conditional on observing the flow y [9]. This is called the posterior probability and is denoted by $p(c_k|y)$. The celebrated Bayes rules gives a way of calculating this value:

$$p(c_k|y) = \frac{p(c_k)f(y|c_k)}{\sum_{c_k} p(c_k)f(y|c_k)} \tag{2.4}$$

where $p(c_k)$ denotes the probability of obtaining class c_k independently of the observed data (prior distribution), $f(y|c_k)$ is the distribution function (or the probability of y given c_k) and the denominator acts as a normalizing constant.

The goal of the supervised Bayes classification problem is to estimate $f(y|c_k)$, $k = 1, \ldots, K$ given some training set x. To do that, Naïve Bayes makes certain assumptions on $f(\cdot|c_k)$ such as independence of f_m's, leading to $f(y|c_k) = \prod_{m=1}^{M} f(f_m|c_k)$, and the standard Gaussian behavior of them. The problem is then reduced to simply estimating the parameters of the Gaussian distribution and the prior probabilities of c_k's. All model parameters (i.e., class priors and feature probability distributions) can be approximated with relative frequencies from the training set. These are maximum likelihood estimates of the probabilities.

The discussion so far has derived the independent feature model, that is, the Naïve Bayes probability model. The Naïve Bayes classifier combines this model with a decision rule. One common rule is to pick the hypothesis that is most probable; this is known as the maximum a posteriori or MAP decision

rule. The corresponding classifier is the function $classify$ defined as follows:

$$classify(y) = \underset{c_k}{\text{argmax}}\ p(c_k|y) = \underset{c_k}{\text{argmax}}\ p(c_k) \prod_{m=1}^{M} f(f_m|c_k) \tag{2.5}$$

2.4.2 K-Means Classifier

In the following, we describe a classification system trained by an unsupervised (clustering) procedure. This approach has been adopted by a number of other works (see Chapter 1) and it's a useful benchmark and performance comparison tool. When dealing with patterns belonging to $\mathbb{R}^n$ we can adopt a distance measure, such as the Euclidean distance. Moreover, in this case we can define the prototype of a cluster as the centroid (the mean vector) of all the patterns in the cluster, thanks to the algebraic structure defined in $\mathbb{R}^n$. Consequently, the distance between a given pattern x_i and a cluster C_k can be defined as the Euclidean distance $d(x_i; \mu_k)$ where μ_k is the centroid of the m_k patterns belonging to C_k:

$$\mu_k = \frac{1}{m_k} \sum_{x_i \in C_k} x_i \tag{2.6}$$

One of the simplest, yet powerful, algorithms for K-clustering is the K-means. This algorithm performs the following steps, given a set A of patterns:

1. *Initialization.* It consists in initializing K centroids, by randomly selecting K different patterns in the data set A. At this stage each cluster is empty.

2. *Main loop.* For each pattern x_i in A:

   Compute the closest centroid:

$$h^* = \arg\max\{d(x_i, \mu_h)\} \tag{2.7}$$

Assign the pattern x_i to the cluster C_{h^*} represented by the centroid μ_{h^*}.

3. *Centroids' update.* For each cluster, compute μ_k by eq. 2.6.

4. *Verify stop condition.* If a predefined stop condition is true then stop; otherwise go to step 2.

Usually the stop criterion is defined as the logical OR of the two following conditions:

- A predefined maximum number of loops (*epochs*) have been performed.
- The average displacements of the centroids between two successive iterations does not exceed a predefined threshold:

$$\sum_{j=1}^{K} \left|\left| \mu_j^{new} - \mu_j^{old} \right|\right| \leq \delta \tag{2.8}$$

A direct way to synthesize a classification model on the basis of a training set S_{tr} consists in partitioning the patterns in the input space (discarding the class label information) by a clustering algorithm (in our case, by the K-means). Successively, each cluster is labeled by the most frequent class among its patterns. Thus, a classification model is a set of labeled clusters (centroids). More than one cluster can be associated with the same label, i.e. a class can be represented by more than one cluster.

Since in the K-means algorithm the number K of clusters in the final partition must be fixed in advance before running the training procedure, K is a critical parameter and it directly represents the structural complexity of the classification model, i.e $\Sigma\{K\}$. Assuming to represent a floating point number with four bytes, the amount of memory needed to store a classification model is $K \times 4 \times (n+1)$ bytes, where n is the input space dimension and assuming to code

class labels with one byte. An unlabeled pattern x is classified by determining the closest centroid μ_i (and thus the closest cluster C_i) and by labeling x with the same class label associated with C_i. It is important to underline that, since the initialization step of the K-means is not deterministic, in order to compute a precise estimation of the performance of the classification model on the test set S_{ts}, for a given value of K the whole algorithm must be run several times, averaging the classification errors on S_{ts} yielded by the different classification models obtained in each run.

Considering this classification system as the core, it is possible to derive a version able to automatically choose the optimal structural complexity, i.e. the number of clusters to be used in the K-means clustering procedure by adopting the s-fold cross-validation. To this aim we can compute the previously defined performance measures $p_{i,j}$ for each value of K belonging to a considered range of reasonable values $[K_{min}, K_{max}]$; these measures can be arranged into a matrix Π of size $(K_{max} - K_{min} + 1) \times s$:

$$\Pi = \begin{pmatrix} p_{K_{min},1} & p_{K_{min},2} & \cdots & p_{K_{min},s} \\ p_{K_{min}+1,1} & \cdots & \cdots & \cdots \\ \vdots & \vdots & \ddots & \vdots \\ p_{K_{max},1} & p_{K_{max},2} & \cdots & p_{K_{max},s} \end{pmatrix} \tag{2.9}$$

Performance of the classification model with structural complexity equal to k is computed as the row average of Π:

$$\pi(k) = \frac{1}{s} \sum_{j=1}^{s} p_{i,j} \tag{2.10}$$

The optimal number of clusters to be used in the classification model is determined as follows:

$$k^* = \arg \max_{K_{min} \leq k \leq K_{max}} \pi(k) \tag{2.11}$$

We define the $p_{i,j}$'s as accuracy measures, i.e. we let

$$p_{i,j} = \frac{\#\text{ of correctly classified patterns in } S_{ts}}{\text{cardinality of } S_{ts}} \tag{2.12}$$

In case of i clusters and of the j-th validation experiment of the s-fold cross-validation. Then, the model having the best generalization capability is the one corresponding to the maximum value in the $\pi(k)$ sequence.

2.4.3 Multinomial Logistic Regression

In statistics, a multinomial logit (MNL) model, also known as multinomial logistic regression, is a regression model which generalizes logistic regression by allowing more than two discrete outcomes. That is, it is a model that is used to predict the probabilities of the different possible outcomes of a categorically distributed dependent variable, given a set of independent variables.

In machine learning, when a classifier is implemented using a multinomial logit model, it is commonly known as a maximum entropy classifier. Maximum entropy classifiers are commonly used as alternatives to Naïve Bayes classifiers because they do not assume statistical independence of the independent variables (commonly known as features) that serve as predictors. However, learning in such a model is slower than for a Naïve Bayes classifier, and thus may not be appropriate given a very large number of classes to learn. But in our case this classifier suites perfectly. In particular, learning in a Naïve Bayes classifier is a simple matter of counting up the number of occurrences of features and classes, while in a maximum entropy classifier the weights, which

are typically maximized using maximum a posteriori (MAP) estimation, must be learned using an iterative procedure.

If the multinomial logit is used to model choices, it relies on the assumption of independence of irrelevant alternatives (IIA), which is not always desirable. This assumption states that the odds of preferring one class over another do not depend on the presence or absence of other "irrelevant" alternatives. For example, the relative probabilities of taking a car or bus to work do not change if a bicycle is added as an additional possibility. This allows the choice of K alternatives to be modeled as a set of K-1 independent binary choices, in which one alternative is chosen as a "pivot" and the other K-1 compared against it, one at a time.

The idea behind the multinomial linear regression, as in many other statistical classification techniques, is to construct a linear predictor function that constructs a score from a set of weights that are linearly combined with the explanatory variables (features) of a given observation using a dot product:

$$score(x_i, c_k) = \beta_k \cdot x_i = \beta_{0,k} + \beta_{1,k}x_{1,i} + \beta_{2,k}x_{2,i} + \cdots + \beta_{M,k}x_{M,i} \quad (2.13)$$

where x_i is the vector of features to be labeled, β_k is a vector of weights (or regression coefficients) corresponding to outcome c_k, and $score(x_i, c_k)$ is the score associated with assigning observation i to class c_k. The predicted outcome is the one with the highest score.

Specifically, it is assumed that we have a series of N observed data points $(x_1, \ldots, x_N)$. Each data point i consists of a set of M explanatory variables, or features, $x_i = (x_{1,i}, \ldots, x_{M,i})$, and an associated categorical outcome c_k, which can take on one of K possible values, that is, there K classes.

To arrive at the multinomial logit model, imagine, for K possible outcomes, running $K-1$ independent binary logistic regression models[6], in which one outcome is chosen as a "pivot" and then the other $K-1$ outcomes are separately regressed against the pivot outcome. If we choose the outcome c_K as the pivot we have that $1 - p(c_k|x_i) = p(c_K|x_i)$ and we can write:

$$\begin{cases} \ln\left(\frac{p(c_1|x_i)}{p(c_K|x_i)}\right) = \beta_1 \cdot x_i \\ \ln\left(\frac{p(c_2|x_i)}{p(c_K|x_i)}\right) = \beta_2 \cdot x_i \\ \dots \\ \ln\left(\frac{p(c_{K-1}|x_i)}{p(c_K|x_i)}\right) = \beta_{K-1} \cdot x_i \end{cases} \tag{2.16}$$

If we exponentiate both sides, and solve for the probabilities, we get:

$$\begin{cases} p(c_1|x_i) = p(c_K|x_i)e^{\beta_1 \cdot x_i} \\ p(c_2|x_i) = p(c_K|x_i)e^{\beta_2 \cdot x_i} \\ \dots \\ p(c_{K-1}|x_i) = p(c_K|x_i)e^{\beta_{K-1} \cdot x_i} \end{cases} \tag{2.17}$$

[6]Binary logistic regression is a type of regression analysis used for predicting the outcome of a binary categorical criterion variable based on one or more predictor variables. The probabilities describing the possible outcome of a single trial are modeled, as a function of explanatory variables, using a logistic function. Since we are in a binary environment we have only two classes c_1 with probability $p(c_1)$ and c_0 with probability $p(c_0) = 1 - p(c_1)$. Let $p(c_1|x_i)$ be the probability of x_i belonging to c_1, then the model can be written as:

$$\ln\left(\frac{p(c_1|x_i)}{1 - p(c_1|x_i)}\right) = logit(P_i) = \beta_1 \cdot x_i \tag{2.14}$$

we can then rewrite the model in terms of in terms of the probability of the outcome occurring as:

$$p(c_1|x_i) = \frac{e^{\beta_1 \cdot x_i}}{1 + e^{\beta_1 \cdot x_i}} \tag{2.15}$$

Using the fact that the K probabilities must sum to one, we find:

$$p(c_K|x_i) = \frac{1}{1 + \sum_{k=1}^{K-1} e^{\beta_k \cdot x_i}} \tag{2.18}$$

We can use this to find the other probabilities:

$$\begin{cases} p(c_1|x_i) = \frac{e^{\beta_1 \cdot x_i}}{1+\sum_{k=1}^{K-1} e^{\beta_k \cdot x_i}} \\ p(c_2|x_i) = \frac{e^{\beta_2 \cdot x_i}}{1+\sum_{k=1}^{K-1} e^{\beta_k \cdot x_i}} \\ \dots \\ p(c_{K-1}|x_i) = \frac{e^{\beta_{K-1} \cdot x_i}}{1+\sum_{k=1}^{K-1} e^{\beta_k \cdot x_i}} \end{cases} \tag{2.19}$$

The unknown parameters in each vector β_k are typically jointly estimated by maximum a posteriori (MAP) estimation, which is an extension of maximum likelihood using regularization of the weights to prevent pathological solutions (usually a squared regularizing function, which is equivalent to placing a zero-mean Gaussian prior distribution on the weights, but other distributions are also possible). The solution is typically found using an iterative procedure such as iteratively reweighted least squares (IRLS) or, more commonly these days, a quasi-Newton method such as the L-BFGS method.

2.4.4 Random Forest

The random forest machine learner developed by Leo Breiman and Adele Cutler, is a meta-learner; meaning consisting of many individual learners (trees). The random forest uses multiple random trees classifications to votes on an overall classification for the given set of inputs. In general in each individual machine learner vote is given equal weight. The forest chooses the individual classification that contains the most votes. Figure 2.4 below is a visual representation of the un-weighted random forest algorithm.

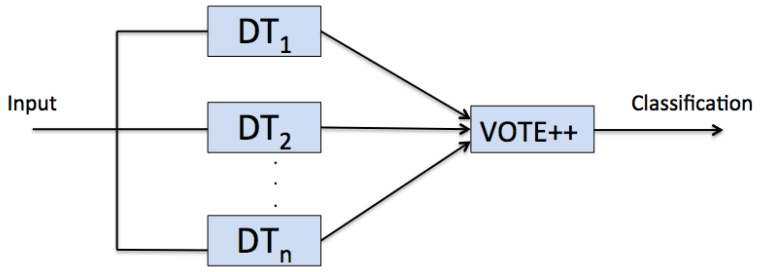

FIGURE 2.3: Meta Learner

Given a dataset of samples (or instances) each consisting of a label and M features, individual random tree machine learners are grown in the following manner:

1. A data set called *inBag* is formed by sampling with replacement members from the training set; this technique is often referred to as "bootstrapping". The number of examples in the inBag data set is equal to that of the training data set. This new data set may contain duplicate examples from the training set. Using the bootstrapping technique, usually one third of the training set data is not present in the inBag. This left over data is known as the out-of-bag data *oob*.

2. If there are M input variables, a number $m \ll M$ is specified such that at each node, m features are selected at random out of the M and the best split (e.g., the attribute maximizing the Information Gain) on these m is used to split the node. The value of m is held constant during the forest growing.

3. Each tree is grown to the largest extent possible. There is no pruning.

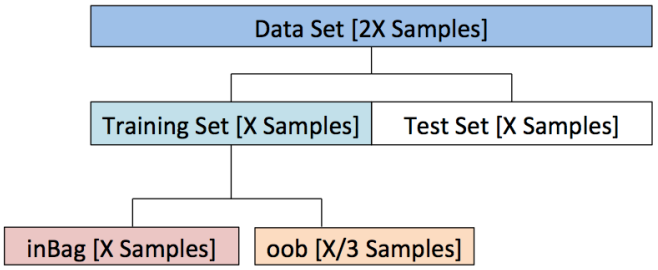

FIGURE 2.4: Sample with replacement

This process is repeated to develop multiple individual random tree learners. After the development of the tree, the oob examples are used to test the individual's trees as well as the entire forest. The average misclassification over all trees is known as the oob error estimate. This error estimate is useful for predicting the performance of the machine learner without involving the test set. This information could be found useful in determining the weights of the individual trees classification in the weighted random forest learner.

Breiman also proved that the forest error rate depends on two things:

- The *correlation* between any two trees in the forest. Increasing the correlation increases the forest error rate.
- The *strength* of each individual tree in the forest. A tree with a low error rate is a strong classifier. Increasing the strength of the individual trees decreases the forest error rate.

Reducing m reduces both the correlation and the strength. Increasing it increases both. Somewhere in between is an "optimal" range of m, which usually is quite wide. Using the oob error rate a value of "m" in the range can quickly be found. This is the only adjustable parameter to which random forests is somewhat sensitive.

2.5 Dataset Creation

To define a reliable application data set, we have used real traffic traces collected at different locations, including:

- NetLab laboratory premises in the campus of "Sapienza", University of Rome;
- Information Engineering department at "Sapienza" University of Rome[7];
- Elsag Datamat main site in Rome[8];
- private homes.

This way we encompass several major kinds of Internet access points: institutional, business and domestic. All of these collected traces consist of real traffic, generated by means of some tools in controlled conditions, so that it is possible to build up data sets with guaranteed label metadata of collected flows (ground truth). A reliable ground truth data set is needed to train supervised machine learning algorithms as the ones used on this work. The controlled traffic generation is a must specifically for collecting traces of application flows carried within encrypted tunnels, i.e. to label each SSH flow with a metadata specifying the service it is carrying among SCP, SFTP and HTTP. A data set for our purposes is composed by a collection of flows, along with metadata per flow, reporting the application layer protocol the flow belongs to, the timestamp of its first packet (T_0), the capture date and location.

[7]Information Engineering Department and NetLab are located in different places within Rome and trace route tests prove that several routers are crossed between the two places

[8]Elsag Datamat is a company of the Finmeccanica group, `www.elsagdatamat.com`

2.5.1 Application Traffic Collection

TCP applications data set is composed by HTTP, FTP-Control Session, POP3 and SSH flows. As for HTTP and FTP-Control Session (FTP-C in the following), we have collected traffic flows originating from clients in the NetLab at DIET Department in the campus of University of Rome Sapienza. The internal network is made up of a switched Ethernet with about thirty stations configured as clients and servers. The internal network is connected to the campus network by means of a 10 *Mbps* link and from there to the public Internet via University backbone. Tens of users are active daily in the NetLab. Traffic traces have been collected on the link between NetLab edge router and the University backbone access router.

By means of automated tools mounted on machines within NetLab, thousands of web pages have been downloaded in a random order, over thousands of web sites distributed in various geographical areas (Italy, Europe, North America, Asia). FTP sites have been addressed as well and control FTP session established with thousands remote servers, again distributed in a wide area. In addition two thousands POP3 flows were collected by capturing traffic generated by different users of NetLab network who handled e-mails during several work days. The artificial traffic (HTTP and FTP-C) as well as POP3 traces have been sniffed from our LAN switch by configuring a mirroring port. We have verified that the TCP connections bottleneck was never the link connecting our LAN to the outside network (access link).

2.5.1.1 SSH Traces

We address different network scenarios using multiple client-server couples to capture SSH traffic. In order to have realistic traces and technology independent implementations of SSH (version 2) protocol, we have used computers

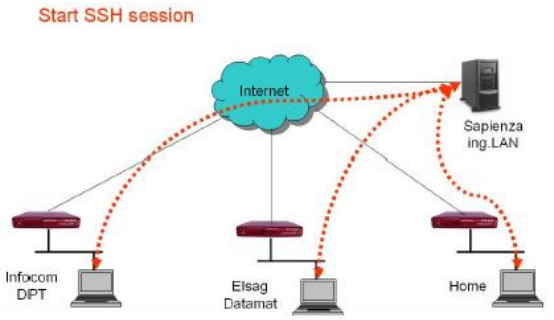

FIGURE 2.5: Platform used to generate SSH traffic: SSH server is inside the University campus network; clients are at University, Elsag Datamat and a private home premise, respectively.

with heterogeneous operative systems, namely Linux and Windows. We simulate SSH connections by connecting three client computers deployed in three different LANs to one server located in a fourth different LAN. As shown in Figure 2.5, client LANs and SSH server have been connected to the Internet by using different geographic links.

We have run the following SSH services: SCP, SFTP and HTTP over SSH. SCP and SFTP are transfer file services natively available on OpenSSH [37]. In particular we downloaded/uploaded files from clients to server using both SCP and SFTP protocols collecting eight thousands flows. HTTP over SSH traces have been collected downloading web pages through SSH tunnels (one SSH tunnel for each HTTP session). This way we have collected four thousands SSH flows carrying HTTP traffic. Throughout these experiments we have considered flows without SSH compression feature. Besides these SSH flows, a tool to automatically generated remote management traffic has been developed. It aim at providing parallel connections to different SSH servers making us able to manage several SSH sessions at the same time. This brought

us to choice Java technology for the development of the system. SSH based management is supported by all operative systems (such as LINUX kernel 2.6, Windows 2003/2008), but it is often used for remotely configure routers and network devices, for example IOS CISCO can be handled by SSH remote connection.

Each operative system is characterized by its configuration commands, therefore an automatic procedure to remotely manage device's console must provide different lists od commands according to the technologies deployed in the network. For this reason the tool provides multi-dictionary mapped on a dedicated database. It stores all information about accounts for logging to devices as well as appropriate commands for the different configurations. To each command is associated a weight pointing out the relevance of the commands itself out of the rest of the complete dictionary. The tool has been set up with constant weight for each command. This means that tools types every commands with the same probability.

Regarding time settings, it is possible to set what we defined as thinking times, enabling tool to send characters every T_{h1} seconds and commands every T_{h2} seconds. T_{h1} and T_{h2} have been set up as equally distributed variables between 0 and 1 s and 0 and 10 s respectively. Setting times this way, put the classification system working in the worst case, in fact it cannot take advantage of the recognition about statistical behavior of the human behavior in thinking and typing commands.

For traffic generation and collection of traces, the tool has run on the platform depicted in Figure 2.6 in which several technology of Servers and Routers have been deployed. The network is managed by several ISP domains and devices are those commonly used in the real networks. The collection of traces, as in the other cases has, been made though a mirroring port of the edge switches.

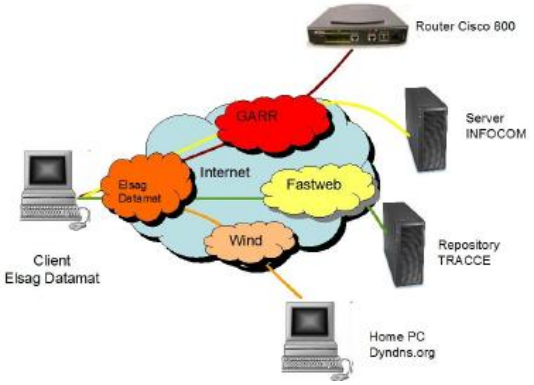

FIGURE 2.6: Platform used to generate SSH remote management traffic.

2.5.1.2 Real Time Application IP Flows: VoIP

The ground truth of the real time applications has been built of artificial VoIP traffic traces. During this phase, we realized three different platforms to get the RTP flows[9]:

- clear mode VoIP;
- the Secure VoIP.

Clear mode VoIP traffic traces have been collected by realizing the network depicted in 2.7 the tools used are:

- commercial softphone (Xlite and Ekiga) to realize call by a PC. Several accounts have been activated to exploit the service of free VoIP calls through the Internet.
- Traditional telephone connected to the PSTN.
- AMTEC IP Phone registered on a SIP public account.

[9]The signaling traffic of the protocol has not been taken into account

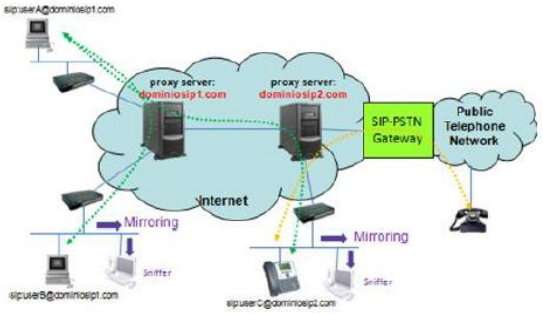

FIGURE 2.7: Clear mode VoIP platform.

We have approached the following call cases:

- Calls between the softphones in different geographic locations of the Internet, collecting RTP traces about the entire VoIP communication.
- Calls between the softphones and the traditional phone, collecting (in the IP phone side of the network) RTP flows coming and going to the PSTN network.

To make a realistic and complete analysis, calls have been run by using all the codecs available, namely: G.729, G.726, GSM, iLBC. Codecs affect the IP packet lengths and the arrival times. The latter highly depends on the network conditions.

The collection of the traces have been made through the edge switch that is very close to the local softphone used for the tests. As in the previous cases the mirror port enabled us to gather pcap traces format.

In order to realize secure VoIP calls, we exploited the VoIP infrastructure of Elsag Datamat. The simplified network diagram is shown in picture 2.8.

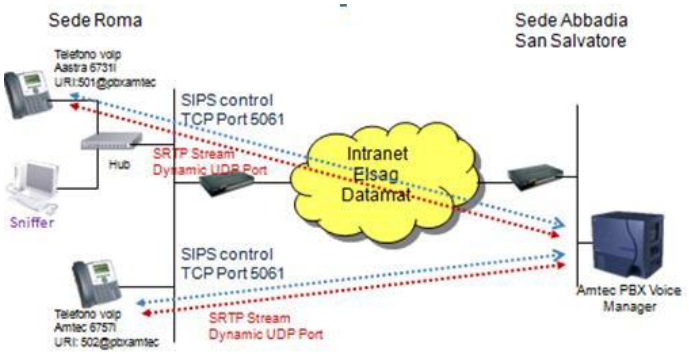

FIGURE 2.8: Secure VoIP platform.

On this network the IP PBX and the IP phones are based on Amtec technology. In particular, the company IP PBX held in the intranet site of Abbadia San Salvatore (Siena, Italy) and IP phones are placed all over the company's intranet. Sites mainly are in Rome, Genoa, Naples etc. All the calls between sites are established by the IP PBX over the company's MPLS and Internet networks.

On this scenario, having the control of the IP-PBX configuration as well as some of the IP Phones located in Rome, enabled us to run secured VoIP calls.

More in details, we were able to access only the IP phones in Rome and we forced the communication through theses phones and outgoing calls toward the IP PBX. Having configured the devices with secure VoIP allowed us to get encrypted calls. SRTP traffic has been captured near of one of the two conversation end points and the IP packets are saved in the tcpdump format.

2.5.2 Traces Pre-processing

A key issue in setting up a useful training data set is pre-processing the collected packet features. From a qualitative point of view, application related information contained in the collected traces should be isolated from TCP and network related effects, e.g. TCP ACK only segments or TCP control segments (e.g. three-way handshake packets), end-to-end round trip times, retransmissions triggered by TCP.

Hence, we remove from each flow $\mathcal{F}$ the following packets: i) first two packets carrying TCP three-way handshake messages: $PK_0 = SYN$, $PK_1 = SYN - ACK$ ii) TCP ACK packets, i.e. those packets carrying only a TCP level ACK and no payload data, that can be recognized because their length is equal to L_{SYN}; iii) retransmitted packets, that can be recognized because their RTX flag is set to one. An example of this first processing phase is given in Figure 2.9.

Let L_{ACK} be the length of the TCP ACK packets (it can be found as $L_{ACK} \equiv \min_{2 \leq j \leq \Gamma - 1} L_j$) and T_0 the time stamp of PK_0, i.e. the SYN packet. Then, the pre-processed data relevant to a given flow $\mathcal{F} = \{\langle d_j, L_j, T_j \rangle . j = 0, \ldots, \Gamma - 1\}$ are: $PK^*_{j-2} = \langle d_j, \lambda_j = L_j - L_{ACK}, \Delta t_j = T_j - T_0 \rangle$ for $j = 2, \ldots, \Gamma - 1$. Packet lengths are so decreased by the TCP+IP header length (including possibly options), so as to return the actual application related data length. Packets turning out to have $\lambda_j = 0$ are discarded (they are just TCP ACK's). Let $\mathcal{P}$ denote the set of indices of pre-processed packet features with positive length. Then the pre-processed flow is $\mathcal{F}^* = \{\langle d_j, \lambda_j, \Delta t_j \rangle \, j \in \mathcal{P}\}$. After tests and analysis of results we set the target value of $P \equiv |\mathcal{P}|$ to strike a convenient trade-off between high classification accuracy and an acceptable classification delay. As our approach is intended to be used in real time, we set a maximum value of P equal to ten. As for SSH traces, we have created a different dataset composed of records extracted from SSH flows only. Since

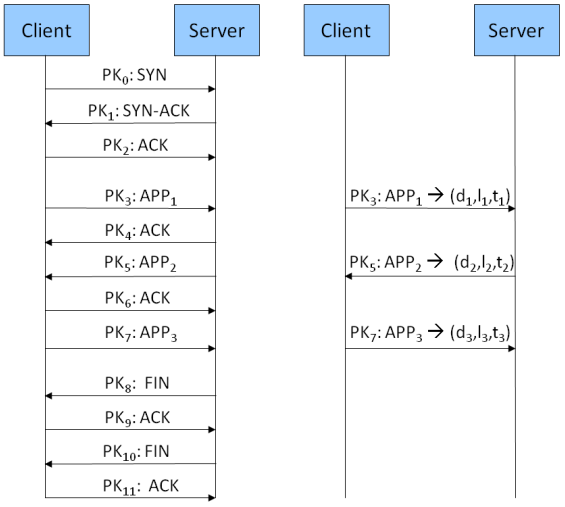

FIGURE 2.9: Preprocessing of a $P = 3$ TCP application flow.

SSH traffic has been generated in controlled way, we know exactly which service is tunneled within each SSH connection. This is the only way we can define a reliable ground truth for encrypted data sets. As we are indeed interested only in those packets that carry the first few encapsulated segments of the tunneled service, we must start collecting packets only after the SSH signaling that triggers the opening of a new forwarding channel. This is a critical point, due to the fact that last packets of SSH handshake phase are encrypted and they can be confused with application data. Moreover SSH offers a wide range of algorithms for encryption and authentication, and this complicates further the detection of the end of SSH handshake phase.

2.6 Performance Metrics

Efficiency can be measured by the average amount of overhead introduced by masking:

$$\mathrm{E}[OH] = \frac{\sum_{i=1}^{M}\sum_{h=1}^{\omega_i} p_{i,h}\left[|\phi(\mathbf{x}_h^{(i)};i)| - |\mathbf{x}_h^{(i)}|\right]}{\sum_{i=1}^{M}\sum_{h=1}^{\omega_i} p_{i,h}|\phi(\mathbf{x}_h^{(i)};i)|} \tag{2.20}$$

where $|\mathbf{u}|$ equals the sum of the components of $\mathbf{u}$, $p_{i,h} = \mathcal{P}(\mathbf{X} = \mathbf{x}_h^{(i)}|\mathcal{A} = \mathcal{A}_i)$ can be estimated from the collected flow for each application and applications have been assumed equiprobable ($P_i = 1/M$). By taking $\mathbf{x}_h^{(i)}$ as the packet lengths we get the *byte overhead*; when times are used, we get the *time overhead*.

Information leakage against flow classification is measured as follows. According to Section 2.1, the adversary defines a classifier TA yielding the application deemed to have generated the observed (masked) flow. Let $\eta_{i,j} = \mathcal{P}(TA = \mathcal{A}_j|\mathcal{A} = \mathcal{A}_i)$ for $i, j = 1, \ldots, M$ and let $\mathbf{H} = [\eta_{i,j}]$ be the flow classifier confusion matrix. Diagonal elements represent success probabilities of the classifier, while off-diagonal elements are mis-classification probabilities. A flow classifier corresponds to a discrete information channel that maps input flows ($\mathcal{A}$ "symbols") into classification decisions (TA "symbols") and is therefore described by the matrix $\mathbf{H}$. By assuming the a priori pdf for application be uniform, i.e. $Q_j = 1/M, j = 1, \ldots, M$, the average mutual information of this "information channel" can be computed as:

$$I(\mathcal{A}; TA) = \log_2 M + \frac{1}{M}\sum_{i=1}^{M}\sum_{j=1}^{M}\eta_{i,j}\log_2\left(\frac{\eta_{i,j}}{\eta_j}\right) \tag{2.21}$$

where $\eta_j = \sum_{r=1}^{M}\eta_{r,j}$. In the following we consider the normalized mean mutual information $\hat{I}(\mathcal{A}; TA) = I(\mathcal{A}; TA)/\log_2 M$, which is just the fraction of

the mean mutual information of the perfect classifier that a real classifier attains. With this definition, $\hat{I}(\mathcal{A}; TA) = 0$ in case of perfect masking, while $\hat{I}(\mathcal{A}; TA) = 1$ for the perfect classifier. We consider also the probability P_{succ} that the adversary correctly classifies the flow application, that can be computed as

$$P_{succ} = \frac{1}{M} \sum_{j=1}^{M} \eta_{j,j} \tag{2.22}$$

Computation of both last metrics requires a matrix $\mathbf{H}$, hence an instance of flow classifier which exploits the masked flow features. We use four classification algorithm (Naïve Bayes, Logistic Model Trees, Random Tree and K-means) offered in the WEKA implementation, which is a machine learning workbench distributed under the GNU General Public License [38]. Each machine learning was trained by feeding it with *masked* flows, so that it can learn to recognize any information that possibly leaks from the set of the masked flow features, that is to say *after* the application of the masking algorithm. This is consistent with the usual security approach where the adversary is granted knowledge of the security algorithm, i.e. the traffic masking algorithm in our case.

Chapter 3

Masking for Perfect Secrecy

In this Chapter we focus on masking techniques providing *Perfect Secrecy*. Initially we analyze the masking of the only packet lengths, first by describing the trivial algorithm with fixed packets length, and after we state an optimization problem to find the full masking algorithm that minimizes the average overhead within the set of ideal algorithms. In the second part we extend to all the traffic features (lengths, times, and direction of packets) the algorithms analyzed. Finally we discuss the results obtained.

3.1 Packet Length Masking

As discussed in previous Sections, packet length is the main statistical feature leaking information about what application originates packets, even if flows are encrypted. We refer to the general model of attack described in Section 2.1 and now we briefly resume it.

We assume a quite general setting, where we can identify origin and destination secure networks (each possibly reducing to a single device), where application endpoints are located (see Figure 3.1). In between there is an

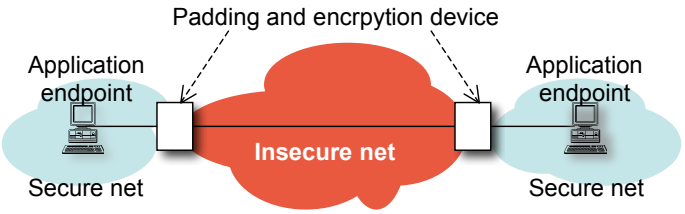

FIGURE 3.1: Scenario for the privacy attack on the packet traffic.

insecure network, where the attacker can observe all flowing packets and attempt to break the privacy of the information flows carried in the insecure network. Confidentiality of packet payload is protected by encryption, but we concede the attacker can identify boundaries of application layer flows, i.e. the attacker can select packets making up a single session of an (unknown) application protocol out of the aggregated packet traffic observed on a network link. Then, the attacker can apply statistical classification to identify the specific application that has originated the flow, even though she can not read into the packet payloads. We want to prevent this attack, specifically the information leakage given by the *packet lengths*. We consider packet padding and/or fragmentation to mask this information.

Since in this Section we are restricting the problem to only packet lengths, the description of the flow entails $\mathbf{X} = [\mathbf{K}, \mathbf{L}, \mathbf{T}]$ expressed in Section 2.1 can be simplified with the relation $\mathbf{X} = [\mathbf{L}] = [L_1, \ldots, L_W]$ where W is the maximum number of packets of a flow.

3.1.1 Fixed Packet Length

Here we consider a practical approach to remove all information leaked by the packet length values, namely fixed length masking. This amounts simply in fragmenting and padding all incoming packets into packets of fixed length L_0.

Masking algorithm with fixed length works as follows. For the r-th packet of a flow, if $L_r < L_0$ the algorithm adds a padding of length $L_0 - L_r$, if $L_r = L_0$ the packet is not modified, whilst if $L_r > L_0$ it fragments the packet into $\lceil \frac{L_r}{L_0} \rceil - 1$ segments of length L_0 plus one segment of length $L_r^* \leq L_0$ to which it will add a padding of length $L_0 - L_r^*$. At the end of masking, the algorithm generates $n = \sum_{i=r}^{m} \lceil \frac{L_r}{L_0} \rceil \leq m \lceil \frac{L_r}{L_0} \rceil$ packets. Practical values of maximum packet lengths is $L_{max} = 1500$ *bytes* for most widespread access networking technology. To keep notation simple, without losing generality, $L_{max} \equiv \ell$. The number n, which is directly proportional to the amount of bytes transmitted for each flow, can be easily exploited by the classifier in order to discover the class that generated the flow. So fragmentation is not sufficient, it is also necessary to mask the total number n of packets corresponding to the original m packets taken into account.This can be done by adding dummy packets of size L_0.

Let us consider two applications $\mathcal{A}_1$ and $\mathcal{A}_2$. Let N_i be the random variable representing the number of packets with fixed length L_0 bytes corresponding to a flow generated by application $\mathcal{A}_i$, $i = 1, 2$. To anonymize the output flow we can add a random number D_r of dummy packets with fixed length L_0 bytes, so that $N_1 + D_1$ and $N_2 + D_2$ have the same probability distribution. In Section 4.1.1 and [39] the following theorem is proved:

Theorem: Let $\pi_i(k)$ be the Probability Mass Function (PMF) of the random variable N_i and $F_i(k)$ the corresponding Cumulative Distribution Function (CDF), for $k = 1, ..., \nu$ and $i = 1, 2$. For any couple of non negative random variables D_1 and D_2 such that $N_1 + D_1 \sim N_2 + D_2 \sim P$, we have $E[P] \geq E[P^*]$ where $F_{P^*}(k) \equiv \min\{F_1(k), F_2(k)\}$, $k = 1, \ldots, \nu$.

By using statistics of the probability distributions of the number of fixed length packets N_i yielded by application $\mathcal{A}_i$, it is possible to compute the minimum number of additional dummy packets required to anonymize each

flow by using the algorithm 1 which is detailed in Section 4.1.1.

3.1.2 Optimum Solution

Also in the following we focus on the case of two applications ($M = 2$) and state an optimization problem that yields a constructive solution for a full masking algorithm $\phi(\cdot; \mathcal{A})$ that achieves minimum overhead in the set of ideal masking algorithms. This optimal full masking algorithm serves as a term of comparison for practical masking, while it is unfeasible to realize, both because of computational complexity and since in principle it requires the entire flow to be available to the masking device to decide upon each message transformation.

Let us assume the sample space (with non null probability) of application i be Ω_i and let $\omega_i = |\Omega_i|$ be the cardinality of Ω_i, for $i = 1, 2$. Outcomes of Ω_i are denoted $\mathbf{x}_r^{(i)}$, $r = 1, \ldots, \omega_i$ $(i = 1, 2)$. For typical application flows, most feature values have null or negligible probability so that $\omega_i \ll |\Omega|$. As found in Section 2.1, full masking entails that the output flow features have the same pdf irrespective of the application that feeds the input of the masking device. To construct the masking algorithm $\phi(\cdot; \mathcal{A})$ we take all ordered couples $(\mathbf{x}_h^{(1)}, \mathbf{x}_k^{(2)})$, with $\mathbf{x}_h^{(1)} \in \Omega_1$ and $\mathbf{x}_k^{(2)} \in \Omega_2$, for $h = 1, \ldots, \omega_1$ and $k = 1, \ldots, \omega_2$. For each couple $(\mathbf{x}_h^{(1)}, \mathbf{x}_k^{(2)})$ we find the optimum masked flow with feature vector $\mathbf{y}_{h,k}$ that the flows in the couple can be mapped to by means of padding (including insertion of dummy messages) and/or fragmentation Optimum here refers to minimization of the overhead required to convert each of the two flows of the couple into the masked flow $\mathbf{y}_{h,k}$. Full masking is obtained by requiring that $\mathcal{P}(\mathbf{Y} = \mathbf{y}_{h,k} | \mathcal{A} = \mathcal{A}_1) = \mathcal{P}(\mathbf{Y} = \mathbf{y}_{h,k} | \mathcal{A} = \mathcal{A}_2) \equiv c_{h,k}$ for all h and k. Then, we

have

$$\mathcal{P}(\mathbf{Y} = \mathbf{y}_{h,k}) = \mathcal{P}(\mathcal{A} = \mathcal{A}_1)\mathcal{P}(\mathbf{Y} = \mathbf{y}_{h,k}|\mathcal{A} = \mathcal{A}_1) + \mathcal{P}(\mathcal{A} = \mathcal{A}_2)\mathcal{P}(\mathbf{Y} = \mathbf{y}_{h,k}|\mathcal{A} = \mathcal{A}_2) = c_{h,k}. \quad (3.1)$$

Optimal full masking is obtained by finding the values of $c_{h,k}$ that optimize the average cost of masking, $z' = \mathrm{E}[D(\mathbf{Y}) - D(\mathbf{X})]$. Here $D(\cdot)$ represents a "cost" measure associated to the flow features. Since the input flow are given, we can reduce the target function to $z = \mathrm{E}[D(\mathbf{Y})] = \sum_{h,k} c_{h,k} D_{h,k}$, where $D_{h,k} \equiv D(\mathbf{y}_{h,k})$ and $c_{h,k} = \mathcal{P}(\mathbf{Y} = \mathbf{y}_{h,k})$. Let $\mathbf{y}_{h,k} = [\lambda_{h,k}]$ in the case of only packet lengths masking. Then, we aim at optimizing byte overhead and we get $D_{h,k} = |\lambda_{h,k}|$.

Summing up, given the costs $D_{h,k}$, the probabilities $c_{h,k}$ are found by solving the following linear optimization problem (*global optimization*):

$$z = \sum_{h=1}^{\omega_1} \sum_{k=1}^{\omega_2} c_{h,k} \cdot D_{h,k} \quad (3.2)$$

subject to constraints:

$$0 \leq c_{h,k} \leq 1 \quad \forall (h,k)$$

$$\sum_{k=1}^{\omega_2} c_{h,k} = p_{1,h}, \quad h = 1, ..., \omega_1$$

$$\sum_{h=1}^{\omega_1} c_{h,k} = p_{2,k}, \quad k = 1, ..., \omega_2$$

where $p_{1,h} = \mathcal{P}(\mathbf{X} = \mathbf{x}_h^{(1)})$, $h = 1, \ldots, \omega_1$ and $p_{2,k} = \mathcal{P}(\mathbf{X} = \mathbf{x}_k^{(2)})$, $k = 1, \ldots, \omega_2$.

We can relate the above optimization problem to the well-known Transportation Problem [40], with the only difference that in our case we have the

quantities to "transport" expressed as fractions of the total amount. The problem at hand is easily solved by the Simplex Method. No efficient solution are known for $M > 3$.

The problem is now reduced to find a constructive way to define the flow $\mathbf{y}_{h,k}$ that minimizes $D_{h,k}$ given the two input flows $\mathbf{x}_h^{(1)}$ and $\mathbf{x}_k^{(2)}$ for all values of (h, k) (*local optimization*). We have set up an exhaustive search to solve the local optimization that turns out to be feasible in case the feature vector is in the order of a dozen components. This gives us sufficient material to use statistical full masking as a comparison benchmark to understand basic trade-offs.

We can summarize the ideal masking algorithm for two applications in the following steps:

1. take as input a flow φ belonging to application 1, with features $\mathbf{x}_{h^*}^{(1)}$ (or: flow ψ belonging to application 2 with features $\mathbf{x}_{k^*}^{(2)}$);
2. draw a random index in the set $[1, \omega_2]$ of value k^* with probability $c_{h^*,k^*}/p_{1,h^*}$ (or: in the set $[1, \omega_1]$ of value h^* with probability $c_{h^*,k^*}/p_{2,k^*}$);
3. transform the input flow φ (or: ψ) into the output masked flow ξ with features $\mathbf{y}_{h^*,k^*}$.

The asterisk highlights that the probabilities $c_{h,k}$ have been obtained by solving the global optimization (3.2) and the table yielding the output masked flow patterns has been filled up by solving the local optimization for any couple of input flows $(\mathbf{x}_h^{(1)}, \mathbf{x}_k^{(2)})$.

3.1.3 Results

In this Section we restrict ourselves to an adversary exploiting packet lengths of the first m packets of each flow, hence $\mathbf{X} = [L_1, \ldots, L_m]$. The results are

Applications Pair	Optimum full masking (FAP)	Optimum full masking (PO)	Fixed length masking
HTTP - SSH	0.3191	0.3605	0.4591
HTTP - FTP-c	0.4088	0.4126	0.5328
HTTP - POP3	0.4351	0.4392	0.5629
HTTP - VoIP	0.3864	0.4126	0.5406
SSH - FTP-c	0.3008	0.3162	0.5378
SSH - POP3	0.3495	0.3715	0.5700
SSH - VoIP	0.2353	0.3546	0.5049
FTP-c - POP3	0.2094	0.2336	0.5302
FTP-c - VoIP	0.2303	0.2752	0.4891
POP3 - VoIP	0.2477	0.2477	0.4892
HTTP over SSH - SFTP	0.2090	0.2248	0.5205

TABLE 3.1: Average amount of overhead introduced by different packet size masking algorithms for various couple of application flows (FAP = Fragmentation And Padding; PO = Padding Only).

obtained with $m = 5$, which is consistent with a real time flow classification target. Albeit restricted, this scenario is enough to highlight interesting issues. Also, packet lengths appear to be the most powerful feature in classification problem.

In Table 3.1 we compare the average byte overhead required by the different masking algorithms. The numbers represent the fraction of the output bytes due to masking overhead (Definition 2.20).

Optimum full masking as defined in Section 3.1.2 comes in two different ways. In the first case, we apply both fragmentation and padding when constructing optimum output flow $\mathbf{y}_{h,k}$ paired with input flows $\mathbf{x}_h^{(1)}$ and $\mathbf{x}_k^{(2)}$ (local optimization). In the second case, only padding (including dummy packets) is allowed. All considered approaches lead to full masking of traffic flows as regards the packet length information. The metric to compare different approaches is therefore the average amount of overhead. As a further comparison

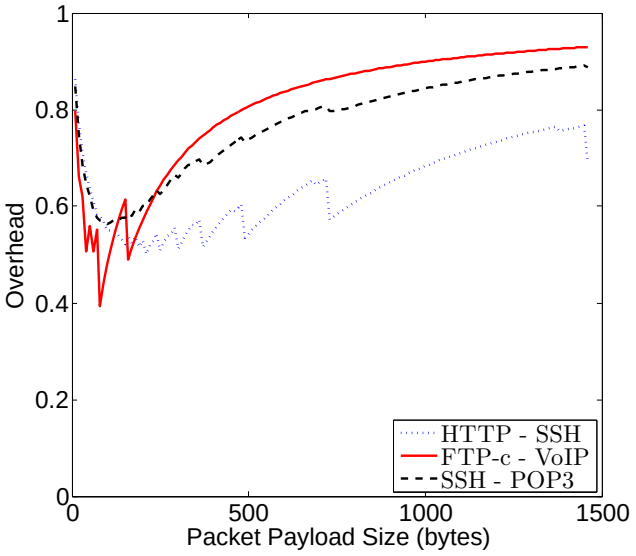

FIGURE 3.2: Comparison between the average overhead obtained by mixing five different couples of applications. The average overhead as defined in eq. (2.20) is plotted as a function of the fixed masked packet length L_0.

we include fixed packet length masking. In Table 3.1 unrestricted optimum full masking is given in the first column, padding only optimum full masking in the second column, while fixed length masking is reported in the third column.

The fixed length L_0 used by the fixed length masking algorithm in Table 3.1 is chosen so as to minimize the average overhead for each masked traffic mix. Figure 3.2 shows the average overhead as a function of packet length L_0 for three different couples of applications that are mixed (HTTP-FTP-c, VoIP-FTP-c, SSH-POP3). There is an optimum choice of L_0, since for very small values the overhead due to fragmentation (additional IP packet headers) is dominant, whereas for large values of L_0 padding is dominant. In any case,

the average amount of overhead is quite large, never less than about 0.5, i.e. about 50% of all bytes sent out by the traffic flow masking device are overhead bytes.

Results in Table 3.1 point out that the amount of overhead can vary significantly depending on the applications mix. In any case, fixed length masking roughly doubles the required overhead with respect to optimal masking, thus showing the optimization of full masking, if possible, can bring significant efficiency gains. An interesting outcome of results in the Table is that optimum full masking constrained to use padding only does not cause a significant increase in overhead compared to the unrestricted optimum full masking, in which we can fragment packets. In a lot of cases, adding fragmentation improves marginally the achieved fraction of overhead. This is a strong argument advocating the use of padding only, although this is not intuitive at first. Main reason is that full masking requires not only masking the length of each packets of the flow but also the amount of bytes of the entire flow.

3.2 All Features Masking

Algorithms seen in the previous Section to mask only the lengths can be easily generalized to all the features in the format presented in 2.1. The two algorithms generalized can be found in the next Subsections.

3.2.1 Fixed Pattern

A much simple approach, but supposedly far from the optimal solution, is *fixed pattern* masking. *Fixed pattern* masking is the generalization of the algorithm *fixed length* masking seen in Section 3.1.1. In general, it means that the input flow, whatever its class, be forced to be framed into a pre-defined pattern with features $\mathbf{y}_0 = [\mathbf{K}_0, \mathbf{L}_0, \mathbf{T}_0]$. Enforcement of these features is obtained

practically as follows. Upon emission of a given burst, the sending application entity has one or more messages. By using fragmenting, padding and delaying those messages are sent according to the desired fixed pattern. If at any given time, an output message must be issued while there are no input bytes to be sent, a dummy message is emitted.

Imposing such a fixed pattern to input flows generated by whatever application is certainly possible and it is guaranteed to raze any possible information useful to the classification adversary and it can be applied message by message. So fixed pattern masking is practical, full masking. The choice of the values of the fixed pattern features $\mathbf{y}_0$ influences the resulting overhead and delay. In general, the choice of $\mathbf{y}_0$ leading to minimal overhead is a multi-dimensional optimization problem, given the overall mix of traffic that it is expected at the masking device. Special cases of the fixed pattern, that simplify its implementation, are obtained by setting a fixed values for all burst sizes, message lengths and message gap times.

3.2.2 Optimum Solution

The generalization of the optimum masking for all the features can be achieved in a trivial way by replacing the vector of the lengths $\mathbf{X} = [\mathbf{L}]$ with the complete vector $\mathbf{X} = [\mathbf{K}, \mathbf{L}, \mathbf{T}]$ in the optimization problem.

Here again *Optimal* full masking is obtained by finding the values of $c_{h,k}$ that optimize the average cost of masking, $z' = \mathrm{E}[D(\mathbf{Y}) - D(\mathbf{X})]$. $D(\cdot)$ represents a "cost" measure associated to the flow features. The target function can be reduced to $z = \mathrm{E}[D(\mathbf{Y})] = \sum_{h,k} c_{h,k} D_{h,k}$, where $D_{h,k} \equiv D(\mathbf{y}_{h,k})$ and $c_{h,k} = \mathcal{P}(\mathbf{Y} = \mathbf{y}_{h,k})$. Let $\mathbf{y}_{h,k} = [\kappa_{h,k}, \lambda_{h,k}, \tau_{h,k}]$. Then, if we aim at optimizing byte overhead we have $D_{h,k} = |\lambda_{h,k}|$; if we are interested in minimizing time overhead (delay), we take $D_{h,k} = \max\{\tau_{h,k}\}$.

Given the costs $D_{h,k}$, the probabilities $c_{h,k}$ are found by solving the same problem of *global optimization* described by the relation 3.2 in Section 3.1.2 solvable by the Simplex Method.

Exhaustive search can be applied to solve the *local optimization* also for the complete set of features. Then, the ideal masking algorithm for all features of the traffic in case of two applications, follows in an identical manner the three steps of the algorithm 3.1.2 for only the packet lengths.

A key limitation of ideal masking algorithms is the requirement to have the entire flow available to decide on masking, before sending out any message to the network. This cannot work for transactional, interactive applications, where a message burst is produced by the application entity based on previously received bursts from the remote entity.

3.2.3 Results

In this Section we report about performance of the algorithms for perfect secrecy. In Table 3.2 we compare optimum full masking and fixed pattern masking[1]. For *fixed pattern*, the same burst size is chosen and independently optimized for each flow direction.

Table 3.3 reports the average overhead for a fixed pattern masking with fixed values of the burst sizes in the two directions (client to server and server to client). The values of the burst sizes are optimized according to the traffic mix, as shown in the Table.

Figure 3.3 shows the average overhead as a function of burst size (equal for all bursts, in both directions) for various application mixes. Figure 3.4 plot the average overhead as a function of the two independent values of burst sizes that can be set in the two opposite directions. The graphs highlight that

[1]In Section 4.2.2 these results will be compared with those obtained with burst by burst masking algorithms.

Application Pair	Optimum full masking	Fixed Burst Size
HTTP - SSH	0.3663	0.5059
HTTP - FTP-c	0.4104	0.5844
HTTP - POP3	0.4233	0.5951
HTTP - VoIP	0.4080	0.5571
SSH - FTP-c	0.3022	0.6082
SSH - POP3	0.3489	0.6285
SSH - VoIP	0.2936	0.5657
FTP-c - POP3	0.1869	0.5263
FTP-c - VoIP	0.2231	0.4872
POP3 - VoIP	0.2700	0.4987
HTTP over SSH - SFTP	0.2752	0.5187

TABLE 3.2: Average overhead introduced by full and fixed burst size masking algorithms for various application mixes.

Application Pair	Opt. Burst Size A $\rightarrow$ B	Opt. Burst Size B $\rightarrow$ A	Minimum Overhead
HTTP - SSH	268	1500	0.4933
HTTP - FTP-c	114	1500	0.5599
HTTP - POP3	114	1500	0.5216
HTTP - VoIP	218	1500	0.5383
SSH - FTP-c	142	168	0.6589
SSH - POP3	105	168	0.6427
SSH - VoIP	142	168	0.5904
FTP-c - POP3	120	273	0.6238
FTP-c - VoIP	200	201	0.4926
POP3 - VoIP	200	100	0.6129

TABLE 3.3: Optimized burst sizes in the two directions and average overhead of fixed pattern masking.

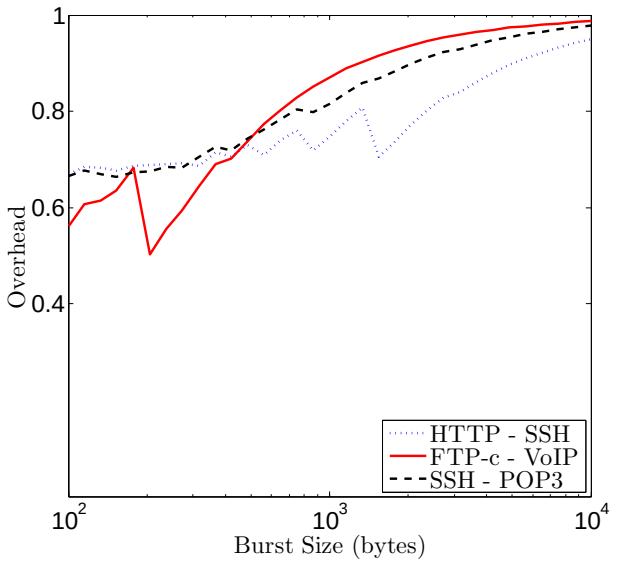

FIGURE 3.3: Fixed pattern masking average overhead of various traffic mixes as a function of the fixed burst size.

there is a big margin of efficiency gain by properly selecting burst sizes, especially if the two dimensional optimization is used, by allowing different value of the burst sizes in the two directions. Even after optimization, overhead values are quite high, but interestingly they are not terribly larger than those of locally optimized additive masking. On the other hand, with fixed pattern masking leakage is stopped completely and implementation complexity is definitely lower than in the much more sophisticated additive masking, that uses statistical information to optimize overhead.

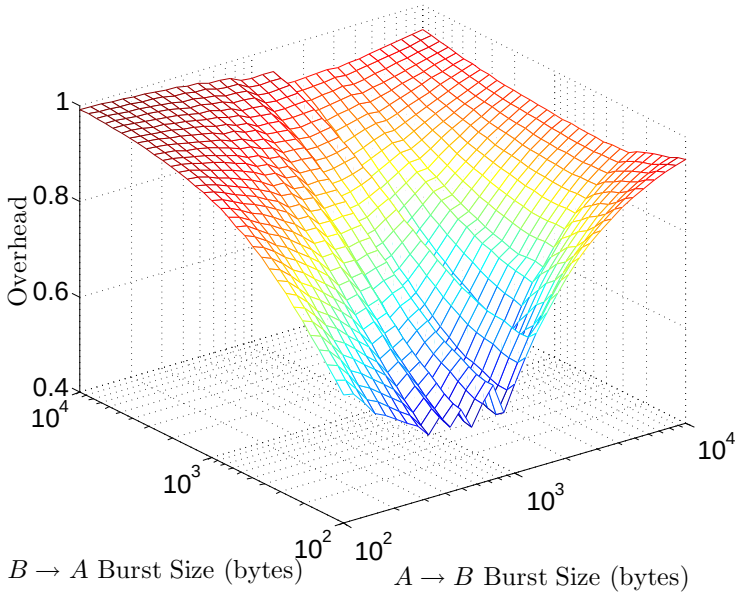

FIGURE 3.4: Fixed pattern masking: average overhead for the mix SSH-VoIP as a function of the burst sizes in the two directions.

Chapter 4

Partial Masking

Complete flow masking for *Perfect Secrecy* can imply massive overhead and/or significant delay, as shown by quantitative examples in Sections 3.1.3 and 3.2.3. In addiction the algorithm for optimum masking cannot work for transactional, interactive applications, where a message burst is produced by the application entity based on previously received bursts from the remote entity.

Next we introduce some practical, partial masking algorithms that allow to reduce amount of overhead and delay at the cost of leaking some information about the content of a flow. The algorithms presented in this Chapter are practically applicable to a message/burst only based on features of previous messages/bursts.

First, as in the previous Chapter, we analyze practical algorithms for packet length masking (Subsection 4.1), and after we generalize the problem to all the feature of a flow (Subsection 4.2). Neither case leads to full masking, since at least correlations among features of different packets/bursts cannot be fully masked.

4.1 Packet Length Masking

The problem of packet length masking is depth in this Section. First we define the optimum padding problem for Marginal Probability Mass Functions (PMFs), independently applicable to the length of each packet (Subsection 4.1.1); after we extend the proposed solutions to the case in which the conditional PMFs are considered instead of just marginal ones, so as to eliminate at least for one-step dependencies (Subsection 4.1.2). In the Subsection 4.1.3 we introduce an algorithm to mask input traffic only partially, so as to knob a trade-off between overhead and degree of masking. At the end the results are presented (Subsection 4.1.4).

4.1.1 Additive Masking for Marginal PMFs

Let us briefly resume some notation. We consider M application layer protocols $\mathcal{A}_i$ for $i = 1, \ldots, M$. As for the packet lengths, we assume application layer entity of each protocol can be characterized by a probability measure. Let $L_r^{(i)}$ be the random variable representing the length of the r-th packet of a flow generated by application protocol i, $r \geq 1$ and $i = 1, \ldots, M$. For any random variable L we let $F_L(n) = \mathcal{P}(L \leq n)$ be the cumulative probability distribution function for $n \geq 0$.

We consider only packet length padding, so that lengths of packets of the anonymized flow are given by $Y_r^{(i)} = L_r^{(i)} + U_r^{(i)}$, where the $U_r^{(i)}$'s are non negative random variables in general. The value of $U_r^{(i)}$ can be a function of $L_j^{(h)}$ for $j \leq r$ and $h = 1, \ldots, M$, which guarantees that the padding algorithm can be run in real time, with minimum delay of padded packets (just processing time delay, no need to wait for following packets). This condition also enables the padding device to be different from the source of packet flow.

Figure 4.1 illustrates a block diagram of the padder. Packet belonging to different application flows enter the edge device connecting the secure network to the public, insecure network. The padder contains tables that give the amount of overhead to be added as a function of the incoming packet length and the application type it belongs to. Such tables can be computed once the probability distributions of the packet lengths of the applications to be mixed have been estimated (see next Section). Then, they are filled and periodically refreshed by a background process that observes incoming traffic (the two upper blocks in the figure, connected with the large white arrow). From analysis of this packet stream, classification of application flows is possible (traffic is assumed to be uncoded in the secure network) and packet length statistics can be estimated.

To read the proper table and apply padding, the *padder box* must know the application the incoming packet belongs to. This information cannot be obtained by classification in real time, since the initial packets of a flow must be released outside to let the application progress. So, when classification is possible in a reliable way, a number of packets have already been released with no padding or a padding compute without knowledge of the correct table to be used. This difficulty can be overcome by means of a cooperating secure network, where some tag is added to application to be mixed, so that they can be recognized by the padder since the very first packet of each flow. A more practical situation can be envisaged in the common case where the secure network reduces to a single host device. Then, padding and background length statistics collection can be carried out by an internal process, e.g. embedded in the operating system. Such a process can obviously know the exact application/service each packet flow belongs to, since they are generated within the same device under the control of a same operating system.

The obtained padded packet is enciphered, so as to protect confidentiality

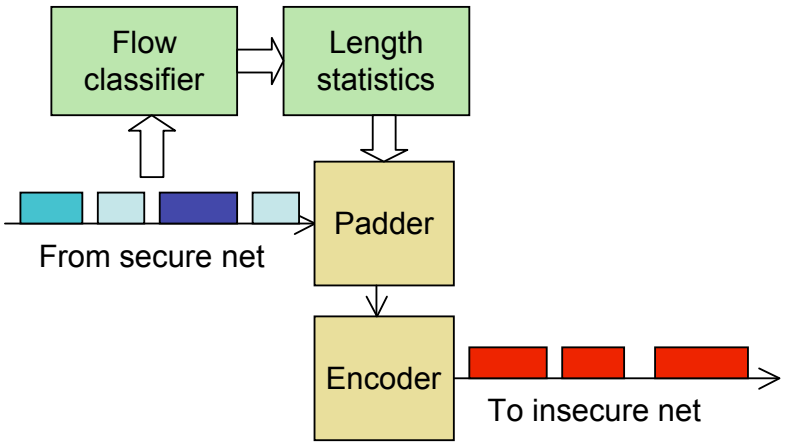

FIGURE 4.1: Block scheme of padder operation in case of trusted/insecure network edge.

of payload and prevent the attacker from removing padding (*encoder box*). At the far end, packets are deciphered, padding is stripped off and clean packets are forwarded to the appropriate application layer endpoint.

In the following, we focus on the padder box. The aim of the padder is to alter packet length so as to confuse a given set of pre-defined application protocols. The key idea is to add a random amount of padding so that lengths of output packets appear as drawn from a *same* Probability Mass Function (PMF) *independently* of the application that has actually generated them.

Let us focus on two application protocols ($M = 2$) and on a specific packet within their respective flows, say the r-th one. We drop the subscript r for the sake of simple notation. Let $a_n = \mathcal{P}(L^{(1)} = n)$ and $b_n = \mathcal{P}(L^{(2)} = n)$ for $n = L_{min}, \ldots, L_{max}$. Practical values of minimum and maximum packet lengths are e.g. 40 *bytes* $\leq L_{min} \leq$ 56 *bytes*, depending on options on IP or TCP headers, and $L_{max} = 1500$ *bytes* for most widespread access networking technology. To keep notation simple, without losing generality, we set $L_{min} =$

1, i.e. the minimum length quantum (e.g. one byte), and $L_{max} \equiv \ell$. We let also $F_a(n) = \mathcal{P}(L^{(1)} \leq n)$ and $F_b(n) = \mathcal{P}(L^{(2)} \leq n)$.

We aim to make packet length series belonging to the two protocols indistinguishable once packets are padded. So, it must be $Y^{(1)} \sim Y^{(2)} \sim Y$. Let $c_n = \mathcal{P}(Y = n)$. We search for a Probability Mass Function (PMF) $\{c_n\}_{1 \leq n \leq \ell}$ among all those satisfying the constraint that only padding be non negative, i.e. $Y^{(i)} = L^{(i)} + U^{(i)}$ with $U^{(i)} \geq 0$, $i = 1, 2$. We show first the following:

Theorem: Let $\{a_n\}_{1 \leq n \leq \ell}$ and $\{b_n\}_{1 \leq n \leq \ell}$ be the PMFs of lengths of packets of two applications. Let $\{c_n\}_{1 \leq n \leq \ell}$ any PMF describing the padded lengths of both applications. Then.

$$F_c(n) \equiv \sum_{j=1}^{n} c_j \leq \min\left\{F_a(n), F_b(n)\right\}, \quad n = 1, \ldots, \ell. \tag{4.1}$$

Proof. Since only padding is allowed, the length of output packets is $Y_1 = L^{(1)} + U_1$ or $Y_2 = L^{(2)} + U_2$, where $Y_1 \sim Y_2 \sim Y$. Therefore, $\mathcal{P}(Y > k) \geq \mathcal{P}(L^{(1)} > k)$ whence $\mathcal{P}(Y \leq k) \leq \mathcal{P}(L^{(1)} \leq k)$. Similarly for the other random variable, $L^{(2)}$. It follows that any output packet length PMF in case of padding must satisfy $\mathcal{P}(Y \leq k) \leq \min\{\mathcal{P}(L^{(1)} \leq k), \mathcal{P}(L^{(2)} \leq k)\}$ or

$$\sum_{j=1}^{k} c_j \leq \min\left\{\sum_{j=1}^{k} a_j, \sum_{j=1}^{k} b_j\right\}, \qquad k = 1, \ldots, \ell \tag{4.2}$$

q.e.d. □

We aim at minimizing the amount of overhead due to padding. Given the PMFs of the padder input packet lengths, this is the same as minimizing E[Y]. We can prove the following:

Theorem: Let $\{a_n\}_{1\leq n\leq \ell}$ and $\{b_n\}_{1\leq n\leq \ell}$ be the PMFs of lengths of packets of two applications. Then $\mathrm{E}[Y^*] \leq \mathrm{E}[Y]$ for any PMF $\{c_n\}_{1\leq n\leq \ell}$ of the r.v. Y under the non negative padding constraint, with the PMF $\{c_n^*\}_{1\leq n\leq \ell}$ of the r.v. Y^* given by

$$F_{c^*}(n) \equiv \sum_{j=1}^{n} c_j^* = \min\left\{F_a(n), F_b(n)\right\}, \quad n = 1, \ldots, \ell. \tag{4.3}$$

Proof. First, we argue $F_{c^*}(n)$ is a proper Cumulative Distribution Function (CDF), if $F_a(n)$ and $F_b(n)$ are. It is non negative, monotonous non decreasing and it attains 1 for $n = \ell$, since both $F_a(n)$ and $F_b(n)$ do so.

Further, we have

$$\begin{aligned}
\mathrm{E}[Y^*] &= \sum_{j=1}^{\ell} j c_j^* = \sum_{j=1}^{\ell} [1 - F_{c^*}(j)] \\
&= \sum_{j=1}^{\ell} [1 - \min\{F_a(j), F_b(j)\}] \\
&\leq \sum_{j=1}^{\ell} [1 - F_c(j)] = \mathrm{E}[Y]
\end{aligned}$$

where last inequality derives from eq. (4.1). q.e.d. □

The PMF $\{c_n\}_{1\leq n\leq \ell}$ is just the target common PMF of the packet length at the output of the padder device to the insecure network. It is the optimum one, i.e. the output packet length PMF with minimum mean value (hence minimum average overhead, given the mean length of input packets) under the constraint that only padding is applied (i.e. no packet fragmentation).

Once $\{c_n\}_{n=1,\ldots,\ell}$ is given, it is possible to compute the PMF of the random overhead U, conditional on the input packet length of the i-th application,

namely $\lambda_{h,k}^{(i)} = \mathcal{P}(U = h|L^{(i)} = k)$ for $h = 0, 1, \ldots, \ell - k; k = 1, \ldots, \ell$ and for $i = 1, 2$. The values $\lambda_{h,k}^{(i)}$ depend on the marginal PMF $\mathcal{P}(L^{(i)} = k)$. As a matter of fact, for $i = 1$ we have

$$\begin{aligned} c_n &= \mathcal{P}(Y = n) \\ &= \sum_{k=1}^{n} \mathcal{P}\left(L^{(1)} = k\right) \mathcal{P}\left(L^{(1)} + U = n|L^{(1)} = k\right) \\ &= \sum_{k=1}^{n} a_k \mathcal{P}\left(U = n - k|L^{(1)} = k\right) \\ &= \sum_{k=1}^{n} a_k \lambda_{n-k,k}^{(1)} \qquad n = 1, \ldots, \ell. \end{aligned} \tag{4.4}$$

The values of $\lambda_{h,k}^{(i)}$ can be computed by Algorithm 1. At step k, we consider the fraction of input packets of length k, i.e. a_k: at the output we have a packet with length n with probability $\left[\sum_{j=1}^{n} c_j - \sum_{j=1}^{k-1} a_j\}\right] / a_k$ (provided this is positive and less than 1). This is simply the probability of the output length be not greater than n minus the probability mass of the output length PMF already "assigned" to input packet of length less than k. Then, the conditional probability that overhead U is no greater than $n - k$ is

$$\begin{cases} \mathcal{P}(U \leq n - k|L^{(1)} = k) = \min\{1, \max\{0, z_{k,n}\}\} \\ z_{k,n} = \dfrac{1}{a_k} \left(\displaystyle\sum_{j=1}^{n} c_j - \sum_{j=1}^{k-1} a_j \right) \end{cases} \tag{4.5}$$

for $k = 1, \ldots, n$ and $n = 1, \ldots \ell$ (as usual it is intended that $\sum_{j=j_1}^{j_2} \equiv 0$ for $j_1 > j_2$).

Let us assume that, for a fixed n, the smallest value of k such that $F_c(n) < F_a(k)$ be k^*; then it is $F_c(n) \geq F_a(k)$ for $k = 1, \ldots, k^* - 1$. Note that it is

Algorithm 1 *PadAlg*: Computation of the PMF of the padding overhead conditional on the input packet length

for $n \leftarrow 1$ to ℓ **do**
  for $k \leftarrow 1$ to n **do**
    $z = 0$
    if $a_k > 0$ **then**
      $z = \frac{\sum_{j=1}^{n} c_j - \sum_{j=1}^{k-1} a_j}{a_k}$
    end if
    $\gamma_{n-k,k} = \min\{1, \max\{0, z\}\}$
  end for
end for
$\lambda_{0,k} = \gamma_{0,k}$
for $k \leftarrow 1$ to ℓ **do**
  for $h \leftarrow 1$ to $\ell - k$ **do**
    $\lambda_{h,k} = \gamma_{h,k} - \gamma_{h-1,k}$
  end for
end for

$1 \leq k^* \leq \ell$ and this is well defined since $F_a(\ell) = 1 \geq F_c(n)\ \forall n$. Then, it is

$$\begin{aligned} z_{k,n} &\geq 1, \qquad k = 1, \ldots, k^* - 1 \\ z_{k^*,n} &= \frac{1}{a_{k^*}} \left(\sum_{j=1}^{n} c_j - \sum_{j=1}^{k^*-1} a_j \right) \in [0, 1) \\ z_{k,n} &\leq 0, \qquad k = k^* + 1, \ldots, n - 1. \end{aligned}$$

Then, we have

$$\begin{aligned} \mathcal{P}(Y \leq n) &= \sum_{k=1}^{n} a_k \mathcal{P}(U \leq n - k | L = k) \\ &= \sum_{k=1}^{k^*-1} a_k + a_{k^*} z_{k^*,n} = \sum_{j=1}^{n} c_j = F_c(n) \end{aligned}$$

The arguments of the proofs as well as the algorithms can easily be generalized to the case of M input PMFs of packet lengths that are to be confused into a single target PMF. The key characteristic of this common PMF is

$$\sum_{j=1}^{k} c_j = \min \left\{ \sum_{j=1}^{k} a_j^{(1)} \, , \ldots , \, \sum_{j=1}^{k} a_j^{(M)} \right\} \tag{4.6}$$

for $k = 1, \ldots, \ell$.

4.1.2 Generalization to Conditional Packet Length PMFs

The Algorithm 1 (*PadAlg*) aims at computing an overhead length PMF used to pad packets from M different application protocols, so that the *marginal* PMF of the r-th packet of each application flow has a resulting length that is drawn from a same PMF, irrespective of the specific application that generated that packet. What we need to compute the target PMF and hence the conditional pad overhead PMFs is knowledge of the PMF of the r-th packet emitted by each application, i.e. $a_k^{(i)}(r) = \mathcal{P}(L_r^{(i)} = k)$, $i = 1, \ldots, M; k = 1, \ldots, \ell; r \geq 1$, where the subscript r of $L_r^{(i)}$ refers to the order of occurrence of the packet inside the flow it belongs to. According to the algorithms defined above, we can compute a padded packet length PMF for each value of r, $\{c_n(r)\}_{n=1,\ldots,\ell}$

This way we neglect correlation information. While marginal distribution of packet length is completely masked, we could expect some information leakage still take place since subsequent packets belonging to a same flow have correlated packet lengths. We can tackle this issue, at least for one-step dependencies, by considering *conditional* PMFs instead of just marginal ones. For the sake of notation, we consider two applications only, the generalization to M being straightforward as done in eq. (4.6). Let $a_k(1) = \mathcal{P}(L_1^{(1)} = k)$ and

$b_k(1) = \mathcal{P}(L_1^{(2)} = k)$; let also

$$\tilde{a}_{k|h}(r) = \mathcal{P}(L_r^{(1)} = k | L_{r-1}^{(1)} = h)$$
$$\tilde{b}_{k|h}(r) = \mathcal{P}(L_r^{(2)} = k | L_{r-1}^{(2)} = h)$$

with $k, h = 1, \ldots, \ell$ and $r \geq 2$. The target padded packet length PMF $\{c_{n|h}(r)\}$ is computed by exactly the same expression as eq. (4.3), except that $\{\tilde{a}_{k|h}(r)\}_k$ and $\{\tilde{b}_{k|h}(r)\}_k$ are fed as input for each given value of h instead of $\{a_k(r)\}_k$ and $\{b_k(r)\}_k$. Analogously, the PMF of the random padding to be applied to a packet of length k belonging to e.g. application 1 is computed from $\{\tilde{a}_{k|h}(r)\}_k$ and $\{c_{n|h}(r)\}_n$ as $\{\tilde{\lambda}_{j|k,h}(r)\}_j = 1, \ldots, \ell - k$ for each given value of h and k. Computational burden is strongly reduced by the typically high correlation found in packet length sequences[1], that imply $\{\tilde{a}_{k|h}(r)\}_k$ is non null only for few values of h.

4.1.3 Tradeoff between Information Leakage and Overhead

Let us return to focus on a specific message position, say the r-th one, within the flow sequence. We drop the index r for the sake of simple notation. Assume we know the pdfs $a_k = P(L^{(1)} = k)$ and $b_k = P(L^{(2)} = k)$, for $k = 1, \ldots, \ell$. Instead of $L^{(1)} + U_L^{(1)} \sim L^{(2)} + U_L^{(2)} \sim \tilde{L}$, partial masking requires that $c_k^{(i)} \equiv \mathcal{P}(L^{(i)} + U_L^{(i)} = k)$ $(i = 1, 2)$ satisfy the following constraint:

$$\sum_{k=1}^{\ell} \min\{c_k^{(1)}, c_k^{(2)}\} \geq q \tag{4.7}$$

where q is a similarity measure, with $q \in [0, 1]$. Complete flow masking is recovered for $q = 1$, that forces $c_k^{(1)} = c_k^{(2)}$, $\forall k$.

[1] This is just another face of the good capability of statistical classifiers found in the literature, as discussed in Chapter 1

Algorithm 2 Partial_Masking(q,**a**,**b**)

$\alpha = 1 - q$
$\{a'_i\}_{k=1,\ldots,\ell} = \{a_k\}_{k=1,\ldots,\ell}$
$\{b'_k\}_{k=1,\ldots,\ell} = \{b_k\}_{k=1,\ldots,\ell}$
for $d \leftarrow \ell - 1$ to 0 **do**
    for $i \leftarrow 1$ to ℓ **do**
        for $j \leftarrow \{i + d, i - d\}$ **do**
            if $1 \leq j \leq \ell$ **then**
                $m = \min\left\{a'_i, b'_j, \alpha\right\}$
                $a'_i = a'_i - m$
                $b'_j = b'_j - m$
                $\alpha = \alpha - m$
            end if
        end for
    end for
end for
assert $\sum_{k=1}^{\ell} c'_k = \sum_{k=1}^{\ell} b'_k = \sum_{k=1}^{\ell} a'_k = 1 - \alpha$
$\{c'_i\} = PadAlg(\{a'_i\}, \{b'_i\})$
$\{c_k^{(1)}\} = \{a_k - a'_k + c'_k\}$
$\{c_k^{(2)}\} = \{b_k - b'_k + c'_k\}$

Our purpose is to find two pdfs $\{c_k^{(1)}\}_{1\leq k\leq \ell}$ and $\{c_k^{(2)}\}_{1\leq k\leq \ell}$ that satisfy eq. (4.7) and minimize the average overhead $\mathrm{E}[OH] \equiv Q_1\mathrm{E}[U_L^{(1)}] + Q_2\mathrm{E}[U_L^{(2)}]$. To solve this problem we have developed algorithm 2. In that algorithm, the function *PadAlg* is the Algorithm 1 and outputs the pdf $\{\gamma_k\}_{1\leq k\leq \ell}$ such that $\sum_{i=1}^{k}\gamma_i = \min\left\{\sum_{i=1}^{k}\alpha_i, \sum_{i=1}^{k}\beta_i\right\}$ $(k = 1, \ldots, \ell)$, for two given pdfs $\{\alpha_k\}_{1\leq k\leq \ell}$ and $\{\beta_k\}_{1\leq k\leq \ell}$.

The algorithm takes as input the two pdfs, $\{a_k\}_{1\leq k\leq \ell}$ and $\{b_k\}_{1\leq k\leq \ell}$, of the message lengths of the two applications we want to mix. Each of them is split in two components according to $a_k = a'_k + a''_k$ and $b_k = b'_k + b''_k$ for $k = 1, \ldots, \ell$. The two components are such that $\sum_{k=1}^{\ell} a'_k = \sum_{k=1}^{\ell} b'_k = q$. Let

a new message of application 1 with length k arrive. With probability a''_k/a_k it is passed onto the enciphering algorithm as it is (no masking), while with probability a'_k/a_k its length is modified according to additive masking so that the conditional output length pdf is $\{c'_k\}/q = PadAlg(\{a'_k\}/q, \{b'_k\}/q)$. This partial masking yields an output pdf of message lengths equal to $\{c_k^{(1)}\}$ given in line 18 of algorithm 2, as we show in the following.

If $u_j^{(1)}(h)$ denotes the conditional probability that padding is $h \geq 0$ bytes, given that the input message belongs to application 1 and has length j, then $c'_k/q = \sum_{j=1}^{k} u_j^{(1)}(k-j)a'_j/q$. Then, the probability of a masked message of length k at the output of the *partial* masking algorithm is $c_k^{(1)} = a_k(a''_k/a_k) + \sum_{j=1}^{k} a_j(a'_j/a_j)u_j^{(1)}(k-j) = a''_k + c'_k = a_k - a'_k + c'_k$, that is just line 18 of algorithm 2. Entirely analogous argument applies to application 2.

The choice of the density portions $\{a'_n\}$ and $\{b'_n\}$ in the rows 4-15 of the algorithm 2 is made so that the amount $|\sum_{k=1}^{\ell} a'_k \cdot k - \sum_{h=1}^{\ell} b'_h \cdot h|$ is minimized. In fact the algorithm gradually constructs them in an iterative way, by excluding the most distant density portions of the two original pdfs, until reaching a set of probabilities with weight q. The components $\{a'_n\}_{1\leq n\leq \ell}$ and $\{b'_n\}_{1\leq n\leq \ell}$ are used to feed *PadAlg*, whose output is the pdf $\{c'_n\}_{1\leq n\leq \ell}$. The resulting pdfs are given in lines 18 and 19 of algorithm 2. It can be checked that these are proper pdfs and that they satisfy eq. (4.7).

4.1.4 Results

According to the scenario defined in Figures 3.1 and 4.1, we consider a collection of traces (ground truth), made up of the ordered sequence of packet lengths of flows belonging to different applications captured and/or generated as commented in Section 2.5.

	No padding	With padding			
		marginal PMFs		conditional PMFs	
m	$\hat{I}(\mathcal{A};TA)$	$\hat{I}(\mathcal{A};TA)$	E$[OH]$	$\hat{I}(\mathcal{A};TA)$	E$[OH]$
1	0.6316	0.0020	0.1275	0.0011	0.1261
2	0.6795	0.0453	0.1034	0.0024	0.0870
3	0.9919	0.0464	0.1234	0.0027	0.1049
4	0.8698	0.2692	0.1762	0.0474	0.0854
5	0.9971	0.1481	0.3717	0.0457	0.3399

TABLE 4.1: Average mutual information of the classifier based on the first m packets of the application flows: scenario with two applications (SSH, POP3).

Four machine learning algorithms described in 2.4 (K-means, Naïve Bayes, Logistic and Random Forest) are used in order to measure the information leakage.

4.1.4.1 Additive Masking

The cumulative probability distribution functions (cpdfs) of the flow first packet for the four considered applications is plotted in Figure 4.2 along with the cpdf of the padded packets, $\{c_n\}$. We are mixing applications with typically short packets (few hundred bytes) such as POP3, SSH and FTP-c, with HTTP, whose packet lengths easily saturate to the maximum 1500 bytes. As a consequence, it is apparent that the probability mass be concentrated around length 100-150 and about 1400-1500. In the light of this, padding overhead is expected to be large.

Results are shown in Tables 4.1, 4.2 and 4.3: the average mutual information $\hat{I}(\mathcal{A};TA)$ (defined in 2.6) associated to the flow classifier K-means and the average fraction of output bytes that are padding overhead (E$[OH]$) are listed as a function of the number m of packets of each flow examined by the classifier according to the overhead definition 2.20.

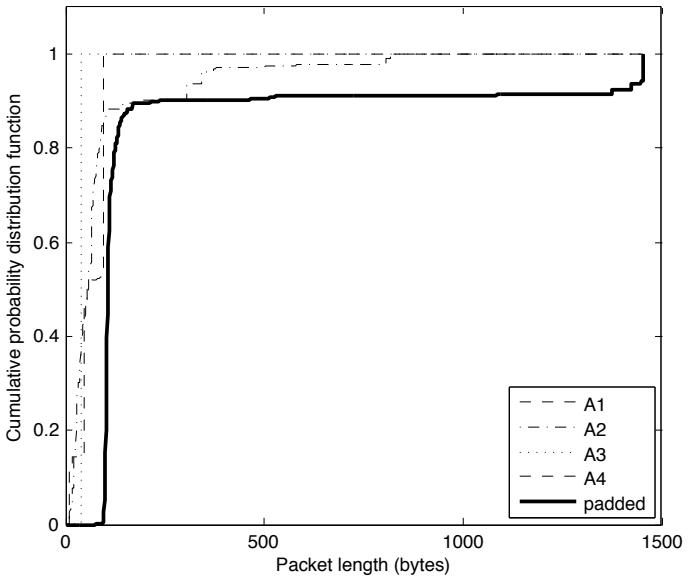

FIGURE 4.2: Cumulative probability distribution function of the packet sizes for the four considered applications ($\mathcal{A}_1 = HTTP$; $\mathcal{A}_2 = FTP - c$; $\mathcal{A}_3 = SSH$; $\mathcal{A}_4 = POP3$) and for the packets padded according to PMF $\{c_n\}$ of eq. (4.6).

	No padding	With padding			
		marginal PMFs		conditional PMFs	
m	$\hat{I}(\mathcal{A};TA)$	$\hat{I}(\mathcal{A};TA)$	E$[OH]$	$\hat{I}(\mathcal{A};TA)$	E$[OH]$
1	0.6096	0.0005	0.3148	0.0005	0.3148
2	0.7938	0.0066	0.5100	0.0027	0.2399
3	0.8267	0.0727	0.6325	0.0710	0.4649
4	0.9093	0.1046	0.6315	0.0818	0.4466
5	0.9115	0.1255	0.6112	0.1131	0.4426

TABLE 4.2: Average mutual information of the classifier based on the first m packets of the application flows: scenario with four applications (HTTP, FTP-c, SSH, POP3).

	No padding	With padding			
		marginal PMFs		conditional PMFs	
m	$\hat{I}(\mathcal{A};TA)$	$\hat{I}(\mathcal{A};TA)$	E$[OH]$	$\hat{I}(\mathcal{A};TA)$	E$[OH]$
1	0.8697	0.0001	0.1295	0.0001	0.1292
2	0.9441	0.0034	0.1570	0.0021	0.0757
3	0.9478	0.0013	0.1183	0.0010	0.0606
4	0.9943	0.0030	0.1463	0.0053	0.1062
5	0.9033	0.0254	0.2256	0.0003	0.1165

TABLE 4.3: Average mutual information of the classifier based on the first m packets of the application flows: two applications tunneled inside SSH connections (HTTP-over-SSH, SFTP).

The padding algorithm is effective in cancelling most of the information provided by the flow classifier, which is otherwise quite successful in detecting origin application, at least when a sufficient number of packets is considered (e.g. $m = 5$). As m increases, a growing amount of information leaks through the padder device, since correlation of the flow packet length sequence are not masked in case of marginal padded packet length PMF or only partially masked in case od one-step conditional padded packet length PMF.

With two application protocols to be mixed up (SSH and POP3, Table 4.1), an almost perfect classifier ($\hat{I}(A;D) = 0.99$ for $m = 5$) is turned into a poor or even an extremely poor tool with random padding based on marginal PMFs (about 0.15 residual average mutual information) or on conditional PMFs (less than 0.05 average mutual information left). Overhead increases as the scope m of the classifier grows, reaching between 34% and 37% of the output traffic. Similar results are found in terms of effectiveness reduction of the classifier in case four applications are considered (HTTP, FTP-c, SSH and POP3, Table 4.2). Overhead is much larger due to the remarkable difference of typical message lengths in HTTP and the other application: the first one tends to exhibit packets close to the maximum 1500 bytes size, the other three protocols

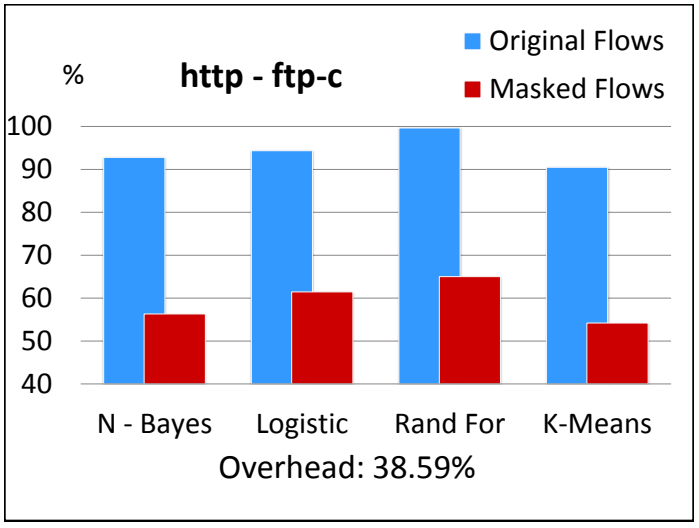

FIGURE 4.3: Comparison between the successful classification probabilities before and after additive masking for the scenario HTTP and FTP-c.

typically send packets between few tens and some hundreds of bytes.

A third different numerical example gives more striking results (Table 4.3). In this case we consider application services tunneled inside SSH connections (so that every packet is entirely encrypted). Random padding as defined in this work is definitely effective in killing classifier capability, e.g. an information leakage that makes the K-means classifier almost perfect for $m = 4$ is largely obfuscated with only about 11% overhead traffic at the ouput of the padding device in case of conditional PMFs and 15% overhead with marginal PMFs. In general, conditional PMF approach has superior performance both in terms of anonymization effectiveness and amount of required overhead.

In the figures 4.3, 4.4 and 4.5 the probability of successful classification P_{succ} (according to the relation 2.22) obtained from the four machine learning before and after additive masking are compared.

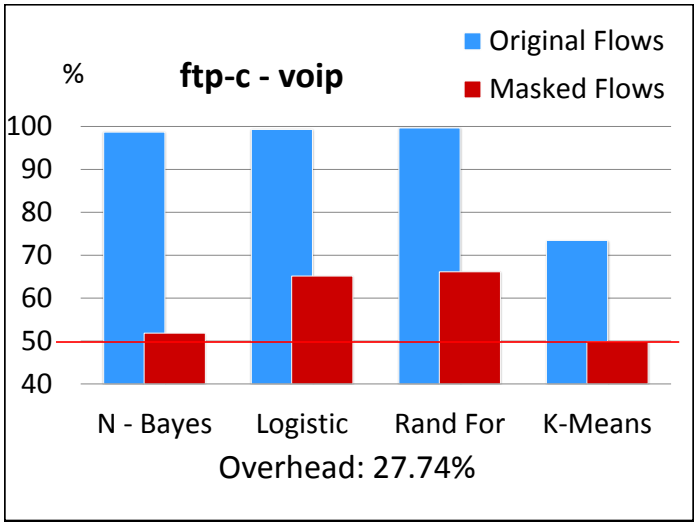

FIGURE 4.4: Comparison between the successful classification probabilities before and after additive masking for the scenario FTP-c and VoIP.

Looking at these results we can observe all the machine learning, which were very effective in the traffic classification, achieve a significant decreasing in performance after additive masking. Due to lengths transformation the P_{succ}'s approach the minimum value (equal to 50% in case of a scenario with two classes).

Random Forest algorithm turns out to be the best among those used, and it obtains the highest P_{succ} in all three scenarios analyzed.

4.1.4.2 Tradeoff between Information Leaks and Overhead

The graphs in Figs. 4.6 - 4.10 show the obtained results for the algorithm described in Section 4.1.3. Figures 4.7 and 4.9 refer to Scenario 1 (SSH and FTP-c flows), while Figs. 4.8 and 4.10 refer to Scenario 2 (HTTP over SSH and SFTP flows, into SSH tunnels). We plot as a function of the masking

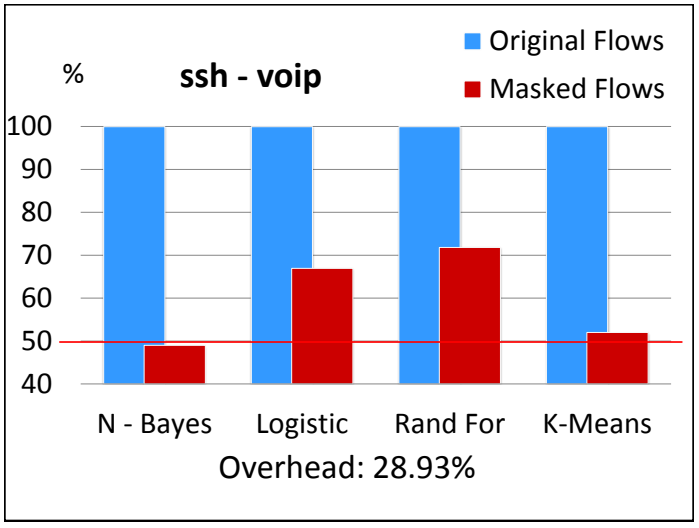

FIGURE 4.5: Comparison between the successful classification probabilities before and after masking for the scenario SSH and VoIP.

rate q the average overhead E$[OH]$ in Figure 4.6 and P_{succ} in Figures 4.7 and 4.8. Figures 4.9 and 4.10 plot the trade-off between the information leakage measured by $\hat{I}(\mathcal{A}; TA)$ and the average overhead. Also here four different classification algorithm have been considered.

The interesting indication of these results is that the amount of overhead cannot be reduced significantly with respect to full masking, if strict leakage requirements are set (q close to 1), yet substantial reduction of overhead with respect to full masking can be attained, if a success probability P_{succ} of about 0.7-0.6 is acceptable (a trivial classifier can attain $P_{succ} = 0.5$). In that case, we can fix $q = 0.8$, that leads to E$[OH]$ about halving with respect to full masking (scenario 2). A smaller gain is obtained in case of scenario 1. As for the trade-off between $\hat{I}(\mathcal{A}; TA)$ and E$[OH]$, the smaller the average mutual information leaked to the adversary, the larger the overhead required. The

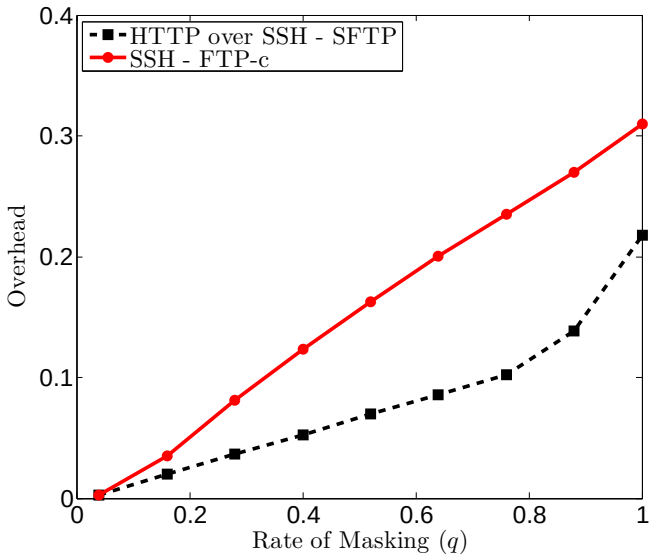

FIGURE 4.6: Average overhead as a function of the masking rate q for the two considered scenarios.

trade-off appears to be much more favorable in case of SSH tunneled application flows (scenario 2). Other tests with different application protocols (e.g. VoIP) have been carried out, yielding similar behavior.

4.2 All Features Masking

In the next, we propose two new practical algorithms (Burst-by-Burst Padding Only and Burst-by-Burst Statistical Additive) applicable to a message/burst only based on features of previous messages/bursts.

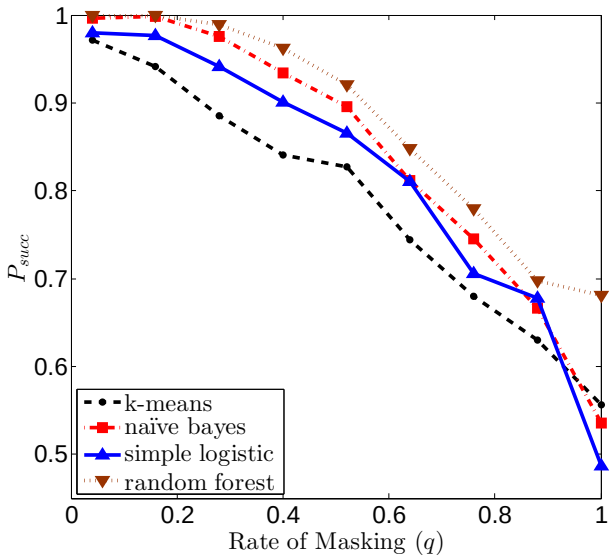

FIGURE 4.7: Probability of a successful classification as a function of the masking rate q for the scenario SSH and FTP-c.

4.2.1 Burst by Burst Masking

The key idea we develop in this Section is to apply the optimized full masking of Section 3.1.2 *burst by burst*, so that the decision on the masking flow can be taken at each endpoint as the traffic flow runs. Given two applications, global and local optimizations can be defined, only restricted to a sub-flow made up of the messages belonging to a given burst, hence with a feature sub-vector $\mathbf{X}_b = [\mathbf{L}_b, \mathbf{T}_b]$, where sub-vector size is equal to the number of messages of the burst. The complexity of the ideal, full masking *within* a burst is limited, since typical burst comprise one or few messages. The overall flow masking is no more full, since correlations across bursts are not taken care of (masking decision is taken burst by burst).

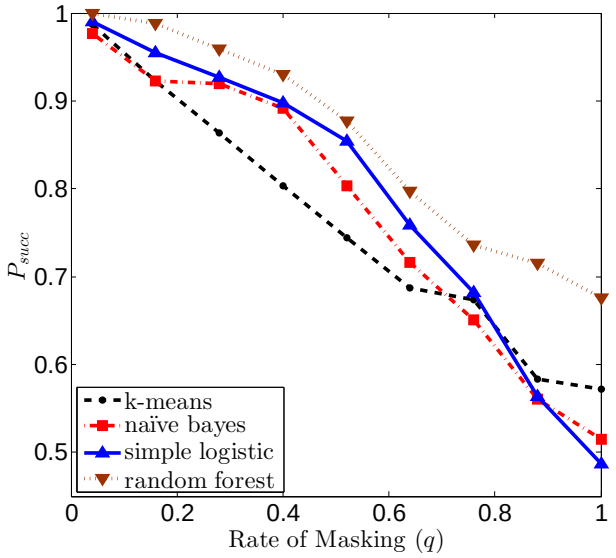

FIGURE 4.8: Probability of a successful classification as a function of the masking rate q for the scenario HTTP over SSH and SFTP.

In general, message padding, fragmentation and dummy messages can be used. If only padding is used, we define the Burst-by-Burst Padding Only (BbBPO) masking algorithm and minimum byte and time overhead is obtained as follows for each burst. Given feature sub-vectors $\mathbf{x}_b^{(i)} = [\mathbf{l}_b^{(i)}, \mathbf{t}_b^{(i)}]$ of application i, $i = 1, 2$, the shortest of the two sub-vectors is padded out with zeros. Then, the output burst feature sub-vector is $\mathbf{y}_b = [\lambda_b, \tau_b]$ with $\lambda_b = \max\{\mathbf{l}_b^{(1)}, \mathbf{l}_b^{(2)}\}$ and $\tau_b = \max\{\mathbf{t}_b^{(1)}, \mathbf{t}_b^{(2)}\}$. Figure 4.11 shows an example of message length masking with padding only for bursts of two applications. Dark shadowed portions of messages are padding bytes.

A different practical masking algorithm can be defined by resorting to *additive masking*. Let $V^{(i)}$ denote a random variable representing a generic feature

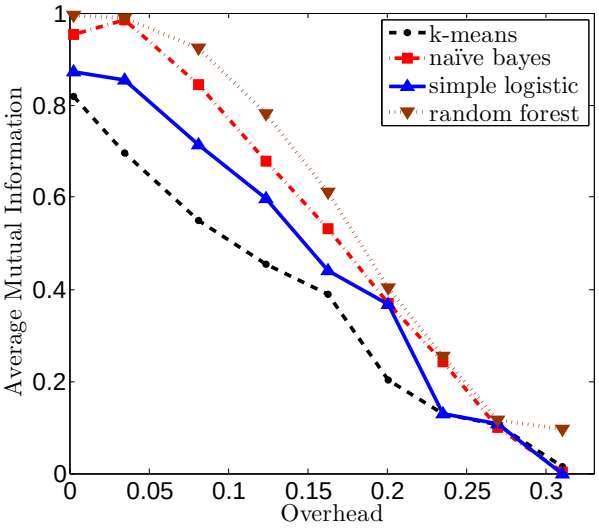

FIGURE 4.9: Trade-off between normalized mutual information and overhead for the scenario SSH and FTP-c.

of a flow of application i ($i = 1, \ldots, M$). We define $F_V^{(i)}(v) \equiv \mathcal{P}(V^{(i)} \leq v)$. With additive masking the masked feature $\tilde{V}$ is built as $\tilde{V} = V^{(i)} + U^{(i)}$, for $i = 1, \ldots, M$, with $U^{(i)}$ a suitable *non-negative* random variable such that the pdf of $\tilde{V}$ is independent of i. In Section 4.1.1 it is shown that minimum overhead additive masking is obtained by choosing $F_{\tilde{V}}(v) = \min\{F_V^{(1)}(v), \ldots, F_V^{(M)}(v)\}$. This choice minimizes $\mathrm{E}[\tilde{V}]$, hence $\mathrm{E}[U^{(i)}]$ for a given value of $\mathrm{E}[V^{(i)}]$, $i = 1, \ldots, M$. Once $F_{\tilde{V}}(v)$ is computed, the pdf of the $U^{(i)}$'s can be calculated; then, given a sample v of $V^{(i)}$, the masking quantity is sampled from the pdf of $U^{(i)}$, say u, and the masked feature value is $v + u$, with $u \geq 0$.

The Burst-by-Burst Statistical Additive (BbBSA) masking paradigm can be applied to the masking of a mix of M applications, whose flows are described

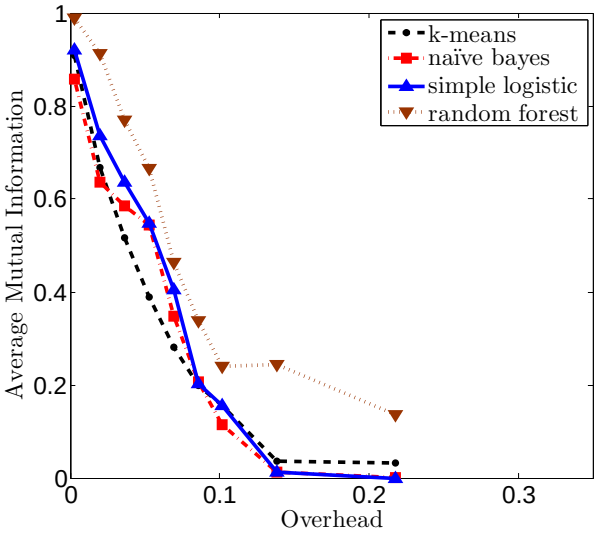

FIGURE 4.10: Trade-off between normalized mutual information and overhead for the scenario HTTP over SSH and SFTP.

by the compact vector features $\mathbf{X} = [\mathbf{B}, \mathbf{\Theta}]$. As for burst number we let $\tilde{N} = N^{(i)} + U_N^{(i)}$, $i = 1, \ldots, M$, so a flows with $N^{(i)} = n$ is padded out with $U_N^{(i)} = u_N$ bursts, whose lengths and epochs are chosen according to the pdfs of the corresponding features. Let $\tilde{n} = n + u_N$ be the resulting number of bursts, with sizes $\mathbf{B}^{(i)} = [B_1^{(i)}, \ldots, B_{\tilde{n}}^{(i)}]$ and gap times $\mathbf{G}^{(i)} = [\Theta_2^{(i)} - \Theta_1^{(i)}, \ldots, \Theta_{\tilde{n}}^{(i)} - \Theta_{\tilde{n}-1}^{(i)}]$ for the i-th application. Then, each burst size and gap time is masked as $\tilde{B}_j = B_j^{(i)} + U_{B,j}^{(i)}$ and $\tilde{G}_j = G^{(i)} + U_{G,j}^{(i)}$, $j = 1, \ldots, \tilde{n}$. Once the sizes of bursts of the masked flow are determined, message length and timing can be defined according to a fixed, pre-defined scheme, independent of the input application flow.

Additive masking as presented above is a practical algorithm, since it can

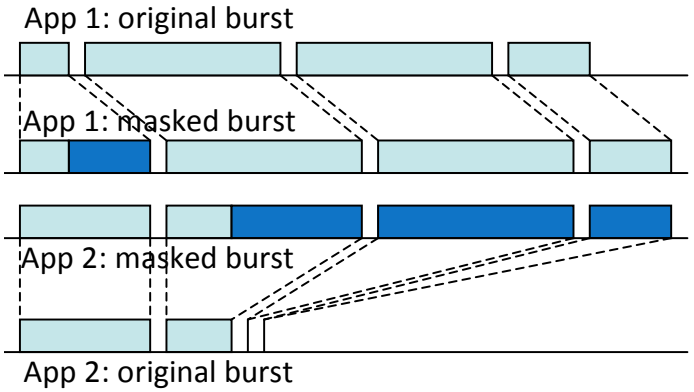

FIGURE 4.11: Examples of message length masking of bursts for two applications.

be applied by each endpoint separately, burst by burst, as they arrive at the endpoint. Knowledge of burst size and gap times statistics is required, but the pdfs of the number of bytes to add to bursts and their delays can be pre-computed, so that the masking device operations can be reduced to table lookup. The device data base comprises M tables for each feature to be masked (e.g. burst lengths). For a given feature V, the table of the i-th application has a number of rows and columns equal to the number of possible outcomes of the feature to be concealed. The entry (u, v) of the i-th table contains the conditional probability $\mathcal{P}(U_V^{(i)} = u|V^{(i)} = v)$, where U_V is the additive masking applied to V.

4.2.2 Results

In this Section we report about performance of practical algorithms described in Subsection 4.2.1. First, we compare optimum full masking, Burst-by-Burst Statistical Additive (BbBSA) masking, Burst-by-Burst Padding Only (BbBPO) practical masking, and fixed pattern masking. We have considered

Application Pair	Opt. full masking	BbBSA Masking	BbBPO Masking	Fixed Burst Size
HTTP - SSH	0.3663	0.3704	0.4428	0.5059
HTTP - FTP-c	0.4104	0.4105	0.4358	0.5844
HTTP - POP3	0.4233	0.4324	0.4479	0.5951
HTTP - VoIP	0.4080	0.3933	0.4255	0.5571
SSH - FTP-c	0.3022	0.3564	0.3799	0.6082
SSH - POP3	0.3489	0.3808	0.3890	0.6285
SSH - VoIP	0.2936	0.3100	0.3824	0.5657
FTP-c - POP3	0.1869	0.2439	0.2710	0.5263
FTP-c - VoIP	0.2231	0.2384	0.3174	0.4872
POP3 - VoIP	0.2700	0.2793	0.3421	0.4987
HTTP over SSH - SFTP	0.2752	0.2823	0.3229	0.5187

TABLE 4.4: Average overhead introduced by practical and fixed burst size masking algorithms for various application mixes (BbBSA = Burst-by-Burst Statistical Additive; BbBPO = Burst-by-Burst Padding Only).

padding only in case of BbBPO since it gives rise to much simpler implementation than in case fragmentation is used and results in Subsection 3.1.3 point out that the overhead penalty is marginal.

Results for the byte overhead, calculated according eq. (2.20), are shown in Table 4.4 in case of compact feature vector $[\mathbf{B}, \mathbf{\Theta}]$ (burst sizes and epochs) and by considering the first 12 bursts for each flow. Remarkably, BbBSA masking incurs a minor penalty as for overhead with respect to optimized full masking. It can be considered as a sort of lower bound of the practical algorithms. Notice that it is possible that $\mathrm{E}[OH]$ for BbBSA be less than for full masking, since BbBSA is a practical masking, hence it is not *full* masking, i.e., it does not fulfill the requirement of removing *any* leakage. Much more overhead is demanded by BbBPO masking and especially by the fixed pattern masking, where a same size of all bursts has been set, with the best choice for each application mix.

Notice that only optimum full masking and fixed burst masking remove any leakage of traffic flow features. Whereas the former is only a theoretical benchmark and cannot be realized, the latter is quite easily implementable and does not even require statistical data. The price to pay for this simplicity is about doubling the overhead with respect to the optimum and also with respect to the BbBSA masking. The appeal of the statistical *practical* masking whose results are reported in the two central columns of Table 4.4 depends on their capability of reducing leakage enough to make classification essentially fail. Since correlations are not taken care of when masking is decided burst by burst, we expect some information is leaked by statistical practical algorithms. Figure 4.12 plots the mutual information $\hat{I}(\mathcal{A}; TA)$ leaked by BbBPO masking for some application mixes as a function of the number of bursts N_B used by the adversary for classification. The classification algorithm is Random Forest.

It is apparent that looking at the first few bursts does not yield a significant amount of information that can be exploited by the adversary. As the number of inspected bursts grows in the order of ten or more, a major leakage is found. Correspondingly probabilities of successful classification range between 0.87 and 0.92 for $N_B \gg 10$.

These findings are confirmed by results in Table 4.5, that refer to BbBPO masking applied to flows by considering up to 50 bursts. The first column shows the values of the average overhead; time overhead in the second column is calculated in the same way by replacing packet lengths with inter-packets gap times. On the third column, the average packet delay introduced by the masking device is displayed. The four columns of the Table 4.6 show the values of P_{succ} obtained with the considered flow classification algorithms.

The results point out that, when we mask all flow features, burst structure, packet lengths and timing information, we get a significant worsening of the traffic load on the network compared with the optimal masking for packet

Application Pair	Byte Overhead	Time Overhead	Average Packet Delay (ms)
HTTP - SSH	0.4363	0.4702	63.310
HTTP - FTP-c	0.4713	0.4239	32.756
HTTP - POP3	0.4408	0.4515	23.879
HTTP - VoIP	0.4523	0.4258	27.313
SSH - FTP-c	0.4210	0.4750	80.345
SSH - POP3	0.4274	0.4814	68.027
SSH - VoIP	0.4198	0.4692	72.255
FTP-c - POP3	0.3992	0.4314	35.263
FTP-c - VoIP	0.3983	0.4156	38.617
POP3 - VoIP	0.4322	0.4529	41.988
HTTP over SSH - SFTP	0.3856	0.4728	50.133

TABLE 4.5: Average byte and time overheads and average delay introduced by BbBPO masking algorithm for various application mixes.

Application Pair	Naïve Bayes	Logistic	Random Forest	K-means
HTTP - SSH	0.6576	0.5303	0.8611	0.6905
HTTP - FTP-c	0.6863	0.5235	0.8779	0.7225
HTTP - POP3	0.5472	0.5094	0.8887	0.7080
HTTP - VoIP	0.5245	0.5112	0.8520	0.6324
SSH - FTP-c	0.5350	0.5900	0.9290	0.5825
SSH - POP3	0.5735	0.7725	0.9297	0.7180
SSH - VoIP	0.6019	0.5128	0.9102	0.6854
FTP-c - POP3	0.6715	0.6445	0.8914	0.6595
FTP-c - VoIP	0.6223	0.6355	0.8922	0.5922
POP3 - VoIP	0.5560	0.6326	0.9022	0.6976
HTTP over SSH - SFTP	0.5490	0.6606	0.8649	0.5712

TABLE 4.6: P_{succ}'s for the four classification algorithms considered.

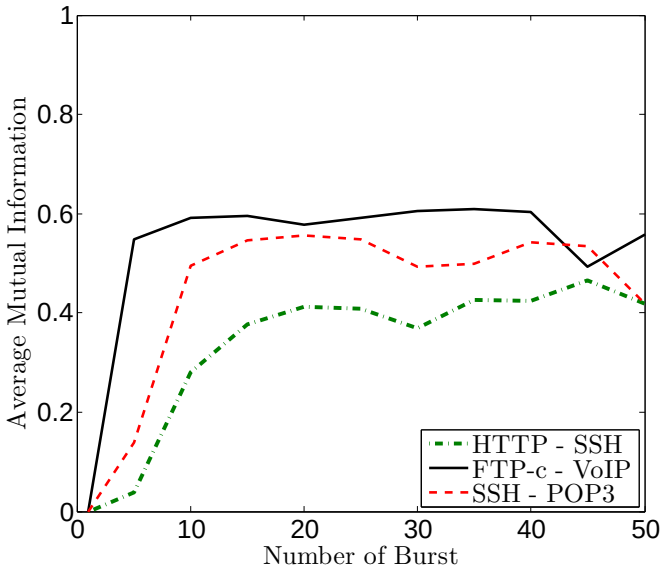

FIGURE 4.12: Average mutual information leaked by BbBPO masking as a function of the number of burst N_B used in the classification for various application mixes.

lengths only. As a matter of fact, the overhead for most cases is more than 40% of the whole output traffic, and in one case it peaks to 47%. Average delay ranges from few tens of ms up to about 80 ms; this is not an issue for most applications but can become critical for VoIP.

In spite of the massive overhead introduced, the key result is that there is still enough leakage for the adversary to be able to classify flow with rather high accuracy, even though a suitable classifier must be chosen. Since intra-burst masking is full (and even optimal), the leakage can be ascribed to correlations among features of different bursts, that are not removed by additive masking. These considerations are also confirmed by Figures 4.13, 4.14 and 4.15

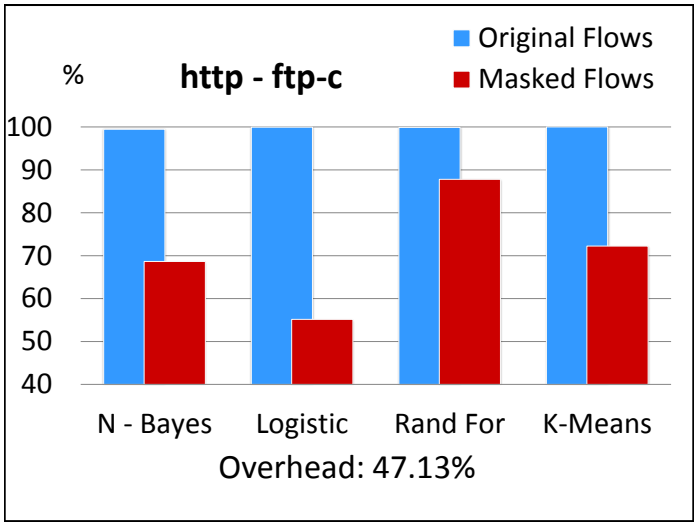

FIGURE 4.13: Comparison between the successful classification probabilities before and after BbBPO masking for the scenario HTTP and FTP-c.

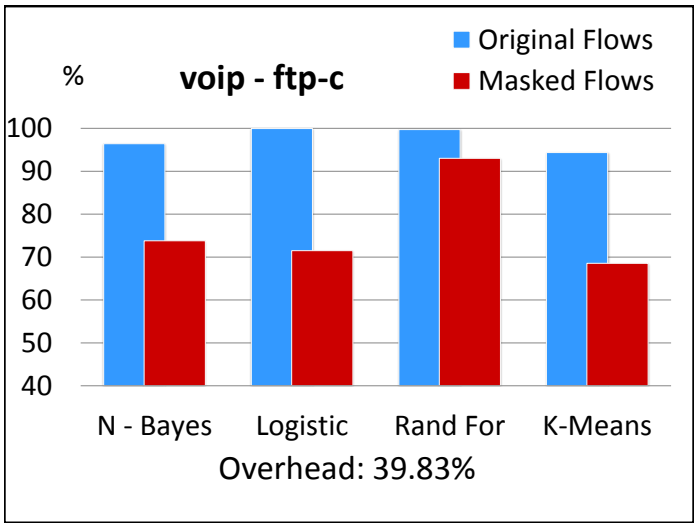

FIGURE 4.14: Comparison between the successful classification probabilities before and after BbBPO masking for the scenario FTP-c and VoIP.

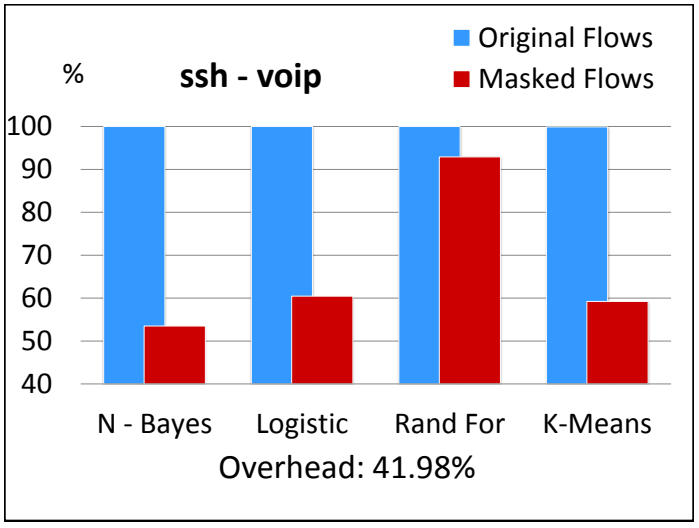

FIGURE 4.15: Comparison between the successful classification probabilities before and after BbBPO masking for the mix SSH and VoIP.

which show the comparison between the information leaks before and after the BbBPO masking. The comparison of the practical masking in Figures 4.7 and 4.8 with $q = 1$ with results in Table 4.6 show that additive masking can be effective if the adversary is limited to observation of the features of a small number of flow packets, as required to attain real time classification, but it fails if the adversary can take the time to observe the flow at length.

Chapter 5

Masking Irrespective of the Application

In this Chapter we aim to approach the problem of applying masking to a packet traffic flow carried in an encrypted channel, irrespective of the application(s) it comes from. Information leakage is stopped by means of traffic feature reshaping, specifically packet lengths and inter-arrival times. We address two contributions: i) it is shown that under the constraint of perfect privacy (namely, cancelation of any information leakage), the optimum masker shapes the protected traffic flow into a fixed length, fixed rate output packet flow; ii) we discuss a heuristic approach to reduce the amount of overhead at the output of the masking device by releasing some controlled information.

As for the Chapter organization, the traffic masking system is introduced in Section 5.1. Section 5.2 is devoted to masking optimization. Numerical examples are presented in Section 5.3, based on real traffic traces.

5.1 Masking Packet Traffic Flows

The reference scenario that we analyze in this Chapter is slightly different from that described in the Chapter 2.

Let us consider two endpoints, denoted with A and B, and a packet flow exchanged between A to B. We focus on $A \rightarrow B$ direction. For ease of language, the term packet is used to refer to data units of the traffic flow, even if they could belong to a layer different from network one. A and B communicate via a secure channel through an insecure network. The adversary can capture packets at will in the insecure network and knows a priori traffic statistics. He can carry out traffic analysis to attack user privacy as discussed in Chapter 1.

Let L_r and T_r be the lengths of the r-packet and the time elapsing between the $(r-1)-$th and the r-th packets as generated at A (r-th inter-packet time). We denote the r-th packet arrival time at A as $t_{a,r}$: then $t_{a,r} = t_{a,r-1} + T_r$. We assume both L's and T's can be modeled as drawn from wide-sense stationary processes, so that the for any r we have $L_r \sim L$ and $T_r \sim T$, with L and T being random variables with at least finite first two moments. From the discussion above we know that the sequences $\{L_r\}_{r\in\mathbb{Z}}$ and $\{T_r\}_{r\in\mathbb{Z}}$ carry information about the traffic flow content, that we refer to generically as a random variable Φ.

The anonymity in the network can be defined after [41] by using the conditional uncertainty of the flow content Φ with respect to eavesdropper's observations, denoted with Ω (packet lengths and inter-packet times of the observed packet flow for each direction). The degree of traffic masking is measured by the normalized equivocation $\alpha = \mathcal{H}(\Phi|\Omega)/\mathcal{H}(\Phi)$, where $\mathcal{H}(L)$ is the average entropy of the random variable L. Perfect privacy corresponds to $\alpha = 1$. In that case, observation of Ω does not yield any additional information about Φ

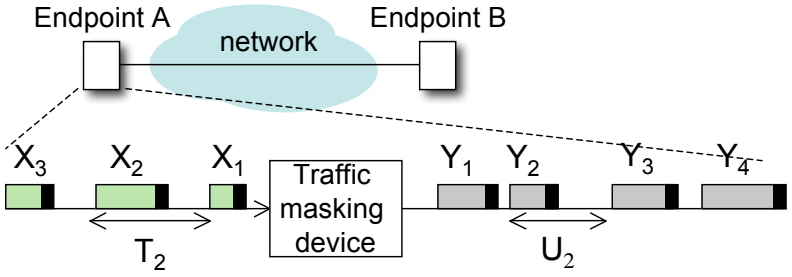

FIGURE 5.1: Sketch of the end-to-end connection with traffic masking: definition of in and out inter-packet inter-arrival times and packet length.

than that provided by the prior probability distribution of Φ. In general, the interpretation of α comes from Fano Inequality: equivocation provides a lower bound to the error probability of the eavesdropper in decoding the flow side information correctly. To cancel any information leakage we replace original packet lengths and inter-packet times with two new sequences, drawn from random process *independent* of the original ones, so that the resulting Ω has no relationship with Φ and it is $\alpha = 1$.

A general scheme of the masking device is given in Fig. 5.1. Let Y_r and U_r be sequences of packet lengths and inter-packet time intervals, chosen for the masked traffic flow. The output packet departure times are denoted as $t_{d,r}$ and $t_{d,r} = t_{d,r-1} + U_r$. The masking device shapes the input original traffic flow so as to impose that the output flow r-th packet has payload length Y_r and it is sent at time $t_{d,r}$.

The logical block structure of the packet masking device is shown in Fig. 5.2. First, the input packet with length L is broken into N new packets of lengths $Y_1, \ldots, Y_N$, whose length are drawn from the output packet length probability distribution function $f_Y(\cdot)$, so that $Y_1 + \cdots + Y_{N-1} < L$ and $Y_1 + \cdots + Y_N \geq L$. The bytes of the input packet are carried by the newly generated

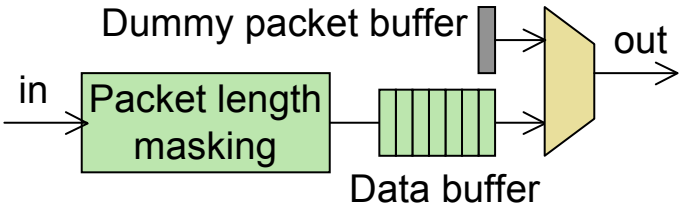

FIGURE 5.2: Masking device with packet length and gap times masking.

fragments. A header of length H is added to each fragment to form the output packet. The header carries the information required to reassemble the original packet at the other end of the secure channel. Those new packets are enqueued in a FIFO buffer. At output departure time $t_{d,r}$, a packet is taken from the data buffer and sent to the output. If the data buffer is empty, a dummy packet is sent to the output, again with length taken from the pdf $f_Y(\cdot)$. Overhead sources aref fragmenting, padding and dummy packets.

In the following, we optimize the choice of the pdfs of the output packet lengths and inter-arrival times of the masking device, by minimizing overhead for a given requirement on the average delay through the device.

5.2 Masking Device Optimization

Let us consider a masking device with output link capacity C, receiving an input packet flow described by inter-arrival times T_r and packet lengths L_r. The masking device shapes the traffic by emitting at the output packets with payload lengths Y_r and a header length H, within time intervals of duration U_r. We assume the average values of all these processes are given. Let $\lambda \equiv 1/\mathrm{E}[T]$ and $\mu_V \equiv \mathrm{E}[V]$ for any random variable V.

The long term average input rate of the masking device is $\lambda\mu_L$, while the output bit rate including overhead is $(H + \mu_Y)/\mu_U$. So, the long term average fraction of overhead at the output of the masking device is $\eta = 1 - (\lambda\mu_L\mu_U)/(H + \mu_Y)$. We aim at minimizing the delay through the masking device for a given value of η. Since the mean delay is a decreasing function of η, this is equivalent to minimizing the overhead η for a given mean delay constraint.

The evolution of the masking buffer can be described by the continuous time random process defined as the buffer content at time t, $Q(t)$. We consider the random sequence $Q_n \equiv Q(t_{d,n}^+)$, $n \geq 0$, that represents the amount of bytes stored in the buffer immediately after the departure of the n-th output packet with payload Y_n. If we assume enough buffering space is provided so that we can neglect overflows, the sequence Q_n obeys a Lindley recursion:

$$Q_n = \max\{0, Q_{n-1} + A_n - Y_n\}, \qquad n \geq 1 \tag{5.1}$$

where $Q_0 = 0$ and A_n represents the amount of bytes arrived at the queue during time interval $(t_{d,n-1}, t_{d,n}]$ with $t_{d,n} = t_{d,0} + U_1 + \cdots + U_n$ and $n \geq 1$.

An explicit form of Q_n can be derived as:

$$Q_n = \max\left\{0, \max_{1\leq\nu\leq n}\left\{\sum_{i=\nu}^{n} A_i - \sum_{i=\nu}^{n} Y_i\right\}\right\} \tag{5.2}$$

For ease of notation, in the following we assume $\sum_{i=a}^{b} \equiv 0$ for $a > b$. Then, it follows

$$\mathrm{E}[Q_n] = \int_{\mathbf{A}_n} f_{\mathbf{A}_n}(\mathbf{a}) d\mathbf{a} \int_{\mathbf{Y}_n} f_{\mathbf{Y}_n}(\mathbf{y}) q_n(\mathbf{a}, \mathbf{y})\, d\mathbf{y} \tag{5.3}$$

where $q_n(\mathbf{a},\mathbf{y}) = \max_{1\le\nu\le n+1}\{\sum_{i=\nu}^{n} a_i - \sum_{i=\nu}^{n} y_i\}$, $f_V(v)$ denotes the probability density function of the random variable V and $\mathbf{A}_n = [A_1,\ldots,A_n]$, $\mathbf{Y}_n = [Y_1,\ldots,Y_n]$. The random vectors $\mathbf{A}_n$ and $\mathbf{Y}_n$ are independent by construction of the masking device, to achieve perfect privacy.

The function $\max_{1\le\nu\le n+1}\{\sum_{i=\nu}^{n} a_i - \sum_{i=\nu}^{n} y_i\}$ is convex with respect to each of the variables y_i. As a matter of fact, let $c_r \equiv \max_{1\le\nu\le r}\{\sum_{i=\nu}^{n} a_i - \sum_{i=\nu,i\neq r}^{n} y_i\}$ and $b_r \equiv \max_{r+1\le\nu\le n+1}\{\sum_{i=\nu}^{n} a_i - \sum_{i=\nu}^{n} y_i\}$. Then, $g(y_r) \equiv \max_{1\le\nu\le n+1}\{\sum_{i=\nu}^{n} a_i - \sum_{i=\nu}^{n} y_i\} = \max\{b_r, c_r - y_r\}$. We can verify that the function $g(z)$ is convex: for any $\alpha\in[0,1]$, z_1 and z_2 we have $c_r - \alpha z_1 - (1-\alpha)z_2 = \alpha(c_r - z_1) + (1-\alpha)(c_r - z_2) \le \alpha\max\{b_r, c_r - z_1\} + (1-\alpha)\max\{b_r, c_r - z_2\} = \alpha g(z_1) + (1-\alpha)g(z_2)$. Since it is also $b_r \le \alpha g(z_1) + (1-\alpha)g(z_2)$. we have $g(\alpha z_1 + (1-\alpha)z_2) = \max\{b_r, c_r - \alpha z_1 - (1-\alpha)z_2\} \le \alpha g(z_1) + (1-\alpha)g(z_2)$.

Since the integrand in eq. (5.3) is convex with respect to each of the variables y_i, $i = 1,\ldots,n$, and the random variables $\mathbf{Y}_n$ are i.i.d. and independent[1] of $\mathbf{A}_n$, we can apply Jensen's inequality and deduce $\mathrm{E}[Q_n] \ge \mathrm{E}[Q_n^D]$ where

$$\mathrm{E}[Q_n^D] = \int_{\mathbf{A}_n} \max_{1\le\nu\le n+1}\left\{\sum_{i=\nu}^{n} a_i - (n-\nu+1)\mu_Y\right\} f_{\mathbf{A}_n}(\mathbf{a})\, d\mathbf{a} \qquad (5.4)$$

with equality iff the random variables Y_i are deterministic. $\mathrm{E}[Q_n^D]$ is the average queue length at $t_{d,n}$ in case of deterministic pdf of the Y's.

The random variables Q_n and Q_n^D converge to proper random variables Q and Q^D, i.e. there exists a stable, stationary limiting state of the queue as $n\to\infty$, if $\mu_A - \mu_Y = \lambda\mu_L\mu_U - \mu_Y < 0$, that is to say the average drift of the queue in negative. Under this assumption, eq. (5.4) yields in the limit as $n\to\infty$ the inequality $\mathrm{E}[Q] \ge \mathrm{E}[Q^D]$, with equality if the random variables Y_i have deterministic pdf.

[1] This is the key to apply Jensen's inequality.

This proves that *the average masking device queue length is minimized by choosing a deterministic pdf with mean $Y_0 \equiv \mu_Y$ for the output packet payload length Y.*

As for the interval lengths U_i, let us consider the workload or unfinished work in the queue at time t, $W(t)$. We consider the random sequence $W_n \equiv W(t_{a,n}^-)$, $n \geq 0$, that represents the amount of workload in the queue found by the n-th arriving input packet. If S_n is the amount of work brought in the queue by the n-th arriving packet, we can write[2]

$$W_{n+1} = \max\{0, W_n + S_n - T_{n+1}\} \tag{5.5}$$

for $n \geq 1$ and $W_1 = 0$. Analogous to the case of Q_n, eq. (5.5) can be made explicit as

$$W_{n+1} = \max_{1 \leq \nu \leq n} \left\{ 0, \sum_{i=\nu}^{n} S_i - \sum_{i=\nu+1}^{n+1} T_i \right\} \tag{5.6}$$

To ease notation, let $\ell_j(\mathbf{L}) = \lceil L_1/Y_0 \rceil + \cdots + \lceil L_j/Y_0 \rceil$; so

$$\mathrm{E}[W_{n+1}] = \int_{(\mathbf{L}_n, \mathbf{T}_n)} f_{\mathbf{L}_n, \mathbf{T}_n}(\mathbf{x}, \mathbf{t}) d\mathbf{x} d\mathbf{t} \int_{\mathbf{U}_n} f_{\mathbf{U}_n}(\vartheta) M_n \, d\vartheta$$

with

$$M_n(\mathbf{x}, \mathbf{t}, \vartheta) = \max_{1 \leq \nu \leq n} \left\{ 0, \sum_{i=\ell_{\nu-1}(\mathbf{x})+1}^{\ell_n(\mathbf{x})} \vartheta_i - \sum_{i=\nu+1}^{n+1} t_i \right\} \tag{5.7}$$

The integrand M_n is convex with respect to each ϑ_i, by an entirely similar argument as in the case of eq. (5.3). By applying Jensen's inequality, it follows that $\mathrm{E}[W_{n+1}] \geq \mathrm{E}[W_{n+1}^D]$, where W^D denotes the workload in case the random

[2] An input packet that finds the queue empty has to wait for the next output sending time, hence it has a nonnull wail equal to the residual time random variable associated to U, thabks to the independence of the times U of the queue state and input process; in that case we should add a term $\mathrm{E}[U^2]/(2\mathrm{E}[U])$; this term is minimized too when U has a deterministic pdf.

variables U_j are deterministic for all $j \geq 1$. If the limiting random variable for $n \to \infty$ exists, namely if the queue is stable, the result applies to the limiting random variable, i.e. $\mathrm{E}[W] \geq \mathrm{E}[W^D]$ with equality iff the random variable U is deterministic with mean $U_0 \equiv \mu_U$.

As a result, the average delay of the masking device queue is minimized, for a given overhead fraction η, if we choose the output shaping with constant amounts of bytes sent out at constant times, as in circuit switching.

5.3 Numerical Results

According to the result in Section 5.2, the optimum masking device under the constraint of perfect privacy consists of fragmenting each input packet into fixed length fragments and sending fragments out to the link at constant rate. Let Y_0 be the length of the output fragment payload, H be the output packet header length, U_0 be the packet inter-departure time at the output of the masking device and C the output link capacity. It must be $(Y_0 + H)/C \leq U_0$. For numerical examples, we assume $H = 20$ *bytes*.

In the following we detail the overhead calculation, define a "relaxed" masking device where some information on the packet length distribution is leaked aiming at a reduction of overhead, and then present numerical results.

5.3.1 Overhead

The average number of bytes per second entering the masking device is $\lambda \mathrm{E}[L]$, while the average number of bytes out of the buffer of the masking device are $(Y_0 + H)/U_0$. So the average fraction of bytes of overhead is $\eta = 1 - \frac{\lambda \mathrm{E}[L]}{(Y_0+H)/U_0}$. The average fraction of overhead due to dummy packets is $\eta_d = 1 - \lambda U_0 \mathrm{E}[\lceil L/Y_0 \rceil]$; the average fraction of padding and fragmentation overhead

per packet are $\eta_p = \frac{Y_0 \mathrm{E}[\lceil L/Y_0 \rceil] - \mathrm{E}[L]}{(H+Y_0)\mathrm{E}[\lceil L/Y_0 \rceil]}$ and $\eta_f = \frac{H}{H+Y_0}$ respectively. It can be checked that $\eta = (1-\eta_d)(\eta_p + \eta_f) + \eta_d$.

The feasible range for η is

$$1 - \frac{\mathrm{E}[L]}{\mathrm{E}[\lceil L/Y_0 \rceil](Y_0 + H)} < \eta \leq 1 - \frac{\lambda \mathrm{E}[L]}{C}$$

provided that Y_0 satisfies $\lambda \mathrm{E}[\lceil L/Y_0 \rceil](H + Y_0)/C < 1$. Once we fix feasible values of η and Y_0, the value of U_0 is given by $U_0 = (H + Y_0)(1 - \eta)/(\lambda \mathrm{E}[L])$.

5.3.2 Relaxed Masking Device

Numerical values of the mean delay and of the average overhead fraction have been obtained by considering a sample IP traffic trace from the CAIDA repository, namely the trace corresponding to the capture file *equinix-sanjose.dirA.20120119-125903.UTC.anon.pcap*. $C = 10$ *Gbps* is the link capacity. The value of $\mathrm{E}[T] = 1/\lambda$ for the considered sample trace is 1.9027 μs. The histogram of packet lengths of the sample trace is shown in Figure 5.3.

This trace comprises 28744877 IPv4 packets (overall about 21.7 IPv4 GB) and 1408949 IPv4 flows.

A strong concentration of most probable values can be noticed. About 40% of packets have close to maximum length, beyond 1400 *bytes*. More than 30% of packets are quite small, with lengths in the order of 100 *bytes* or less. This suggests that a single value of output packet length Y_0 can hardly strike a good compromise, although we know that it is optimum, given the full privacy constraint. If we accept that the fraction of "small" packets as opposed to "long" packets can be observed at the output of the masking device, that is to say if we give up full privacy only to leak this specific information, we can define a dual-length masking device, where input packets are sent to one of two queues depending on their length (see Figure 5.4). Given a threshold value

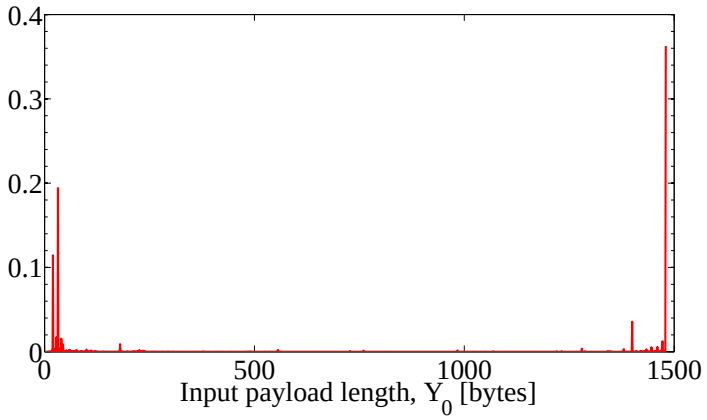

FIGURE 5.3: Histogram of packet lengths for the sample IP packet trace used in the numerical example.

Y_1, if the incoming packet has length $L \leq Y_1$ it is queued in the upper buffer, where the output fixed length is Y_1. If instead the input packets has $L > Y_1$, it is sent to the lower queue, where we set the output packet fixed length to the maximum expected input value, denoted with Y_2. According to the histogram of packet lengths in Figure 5.3. good choices for the output payload lengths are $Y_1 = 100$ *bytes* and $Y_2 = 1480$ *bytes*.

Given the desired output rate C_{out}, the values of the emission intervals U_1 and U_2 of the two queues are chosen so that the average load ϱ on the two queues be the same, i.e., $\lambda q \mathrm{E}[\lceil L/Y_1 \rceil] U_1 = \lambda(1-q)U_2$, where $q = \mathcal{P}(L \leq Y_1)$. Then, $(H + Y_1)/U_1 + (H + Y_2)/U_2 = C_{out}$ and for each given value of C_{out} the corresponding value of ϱ can be calculated.

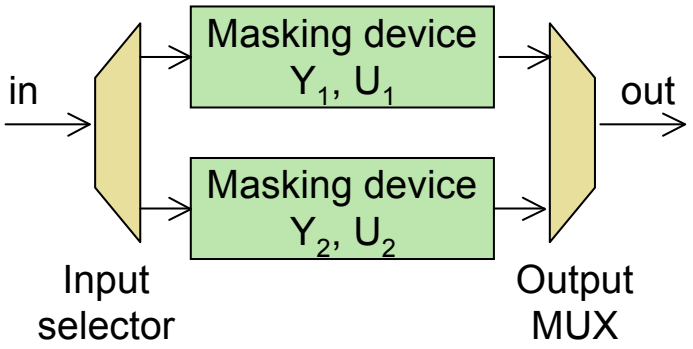

FIGURE 5.4: Logical scheme of the dual buffer masking device.

5.3.3 Simulation of the Masking Device

The workload process of the masking device can be found by simulation, once the sequences of input packet lengths and inter-arrival times are given. For the single buffer device, given the values of Y_0 and U_0, we can write a recurrence for the workload W_n soon after the n-th packet arrival. If $W_{n-1} - T_n$ is positive, then $W_n = W_{n-1} - T_n + S_n$, where $S_n = \lceil L_n/Y_0 \rceil U_0$. If instead $W_{n-1} - T_n$ is negative, a time $U_0\lceil (T_n - W_{n-1})/U_0 \rceil - (T_n - W_{n-1})$ has to elapse before the server can start serving the packet(s) generated by the new arrival. This is the result of the output being rigidly clocked at one output packet per time interval U_0. Summing up, we have:

$$W_n = W_{n-1} - T_n + U_0 \left(\left\lceil \frac{\max\{0, T_n - W_{n-1}\}}{U_0} \right\rceil + \left\lceil \frac{L_n}{Y_0} \right\rceil \right) \tag{5.8}$$

for $n \geq 1$, initialized with $W_0 = 0$.

Trace driven simulations of the masking device have been carried out with the packet length and inter-arrival times extracted from the CAIDA trace described above. The CAIDA trace is made up of carried traffic over a link,

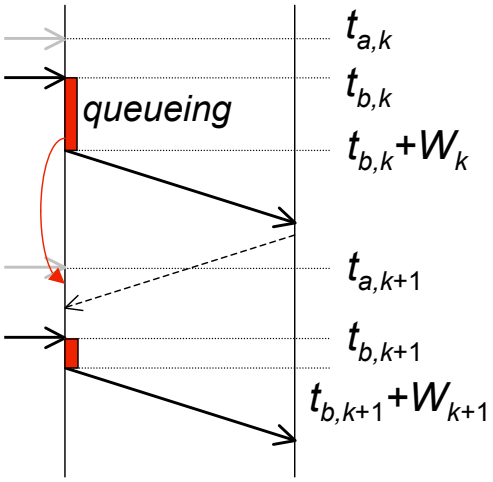

FIGURE 5.5: Example of correction of arrival times of packets (timestamps as observed in the CAIDA trace) of a same application flow to account for the effect of the masking device in the flow loop.

so that we have to account for the effect of the insertion of the masking device buffer on packet inter-arrival times. This effect arises because of conversational applications, where a packet in one direction triggers a response and the next packet in that direction can only arrive after the response has been received. If a delay element is introduced in the data path, the inter-arrival times between consecutive packets belonging to a same application flow are affected by the delay. Application flows are identified on the basis of the values of the IP source address, IP destination address, source port and destination port, provided the gap between two consecutive packets be less than $T_{thresh} = 10$ *sec*.

For a packet flow, let P_r the r-th packet of the flow and $t_{a,r}$ its original timestamp in the measured trace ($r \geq 1$). Let W_r be the delay of the r-th packet through the masking device. Then, the arrival time of the subsequent packet of the same flow is delayed to the corrected arrival time $t_{b,r}$, to account

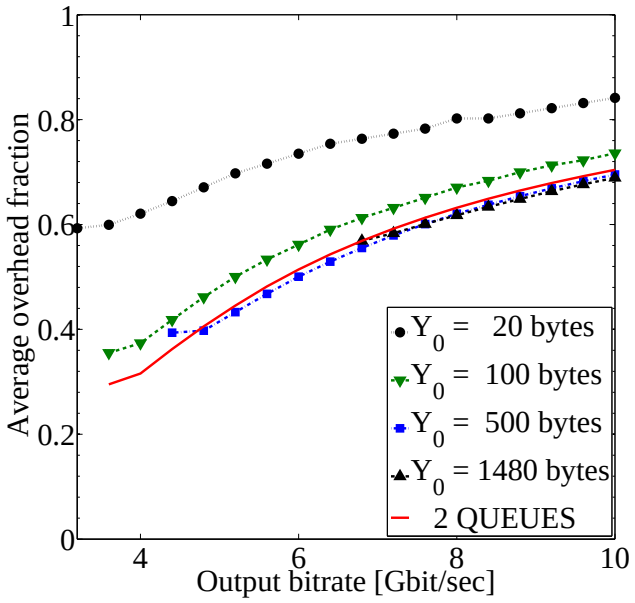

FIGURE 5.6: Performance of masking: average overhead vs. output bit rate.

for the effect of the masking device on the traffic, that is we let $t_{b,r+1} = t_{b,r} + t_{a,r+1} - t_{a,r} + W_r$ (see Figure 5.5).

5.3.4 Performance Results

Figures 5.6 and 5.7 plot respectively the average overhead and the average delay through the masking device as a function of the output bitrate for some values of Y_0. The general behaviour of these curves is characterized by a monotonic trend. The overhead for each curve starts from a minimum value between 0.3 and 0.4 and increases until reaching a value between 0.6 and 0.7. Only for $Y_0 = 20$ *bytes* we have a curve significantly higher compared to the other ones, with overhead between 0.6 and 0.85. The smallest overhead, equal to 0.3, is achieved by the dual buffer masking device.

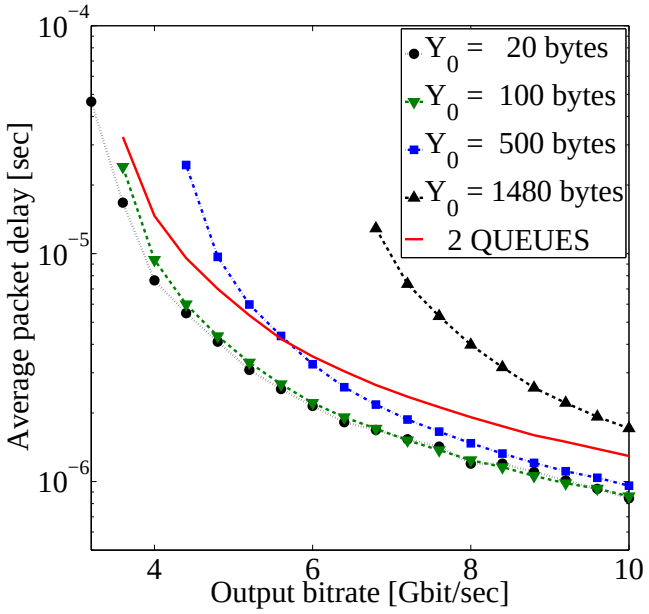

FIGURE 5.7: Performance of masking: average delay vs. output bit rate

The trade-off between the average delay and the overhead is shown in Figure 5.8, for some values of Y_0 and for the device with dual buffer. Samll delays can be obtained at the price of very large overhead values. If a masking delay up to about $20\mu s$ can be introduced, the dual buffer masking device leads to overhead values of 0.3, i.e., 30% of the output rate is masking overhead. This is not as bad a result, considering that privacy protection is essentially complete, the only leaked information being the fraction of the overall input traffic consisting of short packets.

Figures 5.9, 5.10 and 5.11 shows how the overhead is distributed among padding, fragmentation and dummy packets for $Y_0 = 100, 1480$ *bytes* and for the dual buffer device. When $Y_0 = 100$ *bytes* we can observe that the overhead is mainly dominated by dummy packets, to which a good percentage

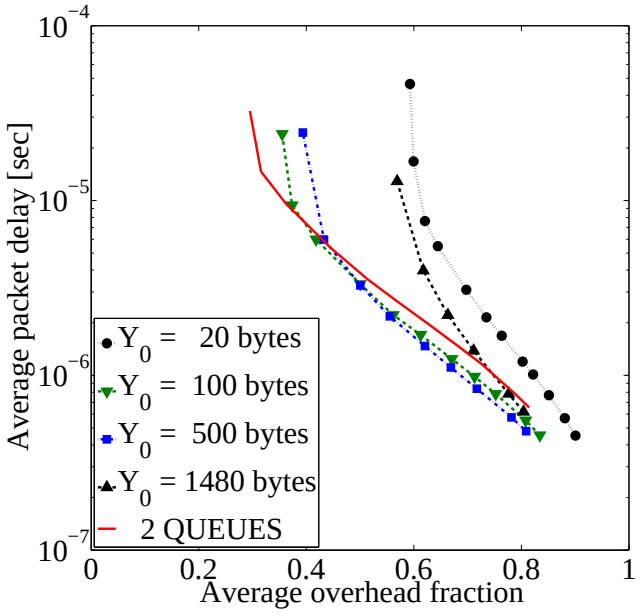

FIGURE 5.8: Performance of masking: masking device trade-off between average delay and average overhead.

of fragmentation overhead is added. With $Y_0 = 1480$ *bytes* the amount of overhead due to padding is very high and significantly increases the total overhead, while no fragmentation is required. Figure 5.11 shows how the dual buffer device allows to obtain a lower overhead by almost eliminating the fragmentation overhead.

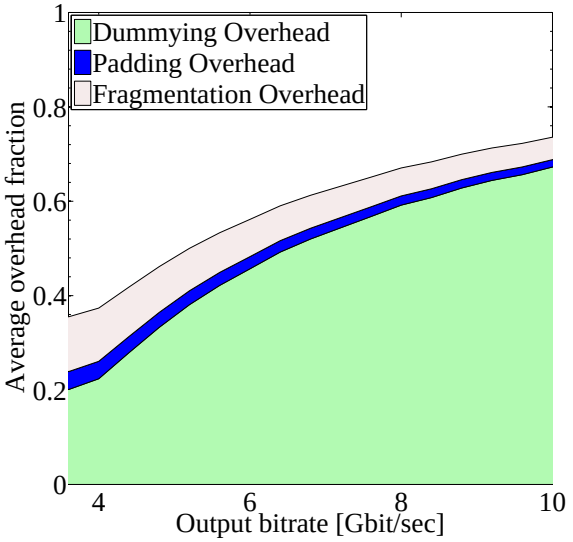

FIGURE 5.9: Contribution of different overhead sources: $Y_0 = 100$ *bytes*.

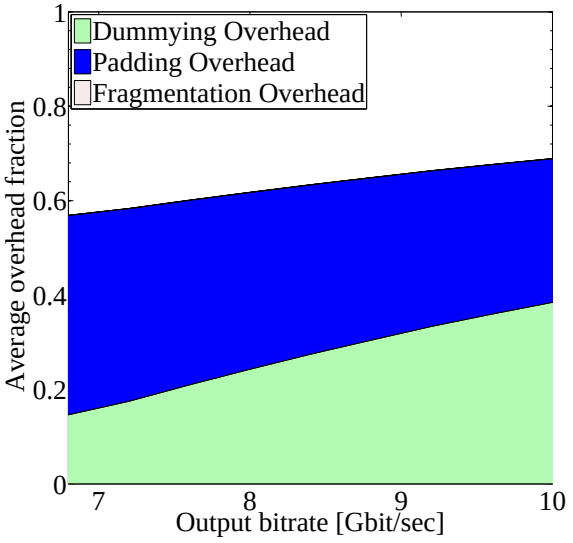

FIGURE 5.10: Contribution of different overhead sources: $Y_0 = 1480$ *bytes*.

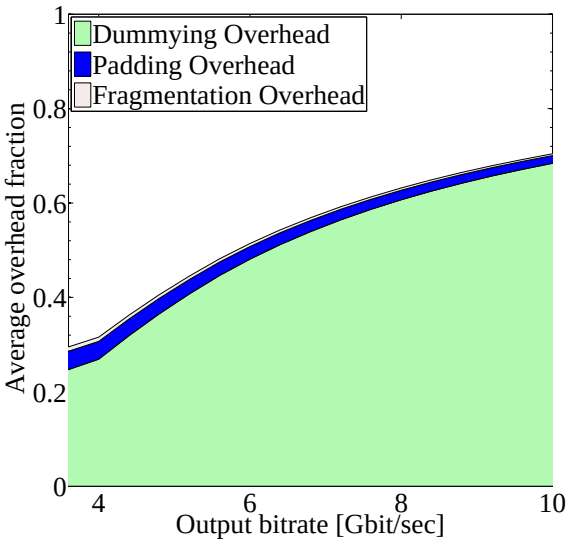

FIGURE 5.11: Contribution of different overhead sources: double buffer masking device.

Chapter 6

Final remarks and outlook

An increasing amount of networking research is focusing on traffic flow classification, since it can be useful for enforcement of security policies and traffic filtering, or it can support quality of service mechanisms. In particular, several methods of classification based on statistical analysis of traffic patterns and machine-learning techniques have been proposed and analyzed. Statistical classification takes some features of the flow packets (e.g. packet lengths, inter-arrival times, direction) and exploits these information to infer which application or service is running those packets, among a set of possible alternatives. Furthermore, based on packet features and possibly other context informational, it is shown that other type of privacy breaking of supposedly end-to-end secure channels exist; as matter of example web pages identification, language/phrase detection in VoIP communication have been successfully demonstrated against encrypted channels.

In this Thesis I aimed at investigating a complementary viewpoint, namely *protection* of privacy against traffic analysis. The same study highlighted how much effort and how complex it is to obfuscate the information leaked by traffic features. We defined the security model pointing out what the ideal target of

masking is, and we defined the some optimized and practically implementable masking algorithms, yielding a trade-off between privacy and overhead/complexity of the masking algorithm.

6.1 What Have We Achieved?

The first interesting finding is that *Optimum Masking* in Chapter 3, even if it is not feasible for transactional, interactive applications, is useful to characterize the full concealment case and it offers a theoretical bound on the amount of overhead necessary to achieve perfect secrecy. It also has shown that fragmenting does not achieve significantly better performance than simple padding as far as overhead-obfuscation trade-off is concerned.

Considering *Practical Masking*, numerical results, based on measured Internet traffic traces, point out that a basic distinction must be done between *real time* adversary and *off line* adversary, observing features of an extended segment of the flow. In the former case, even if some information useful to the adversary leaks when relaxing to practical masking from full masking, still it appears that classification is essentially impaired (success probabilities of best algorithms we could find are below 0.6 in case of two applications). This is tied to the limited number of features used by the adversary, compelled by the need to decide on a class of the observed masked flow in real time, as the flow starts. Things turn out to be completely different if the adversary can take time to classify the flow and observe its features over several bursts, in the order of tens. In that case practical masking, though canceling any information useful for classification inside each burst, cannot remove entirely correlations across features of different bursts[1]. So, statistical, practical masking, even though it minimized overhead burst by burst, still introduces a considerable

[1]This is similar, though much more complex, to mono-alphabetic ciphering, when attacked by exploiting language redundancy, hence correlation.

amount of overhead while failing to protect privacy against an off line adversary. Another question concerns the amount of data requested in advance by the masking device, in order to carry out an accurate process of obfuscation. In fact, it requires knowledge of estimates of the probability density functions for all used features of the considered M applications. As the duration of a flow grows up, the amount of data required becomes very high and quite hard to estimate reliably.

Our numerical investigation gives merit to simpler masking approaches, that give up to global or local optimization leveraging on statistical masking and resort instead to rigid, fixed pattern masking. While increasing overhead with respect to statistical masking, as expected since no optimization is attempted with fixed masking, yet that approach removes any information that could be exploited by the classification adversary, preserves implementation simplicity, and overhead price is not terribly greater than that entailed by optimized solutions (within a factor of 2 from the full, ideal optimized masking), at least in cases we have experimented.

A different and potentially promising approach we had pursued in Chapter 5 is the masking irrespective of the application(s) it comes from. Given full privacy is required, the optimum masking device shall shape the packet flow so that fixed length packets at fixed times are sent through the insecure network. In this context dummy packets are the major source of overhead, due to input traffic burstiness. Mitigation of overhead can be obtained by exploiting the input packet length pdf and let some minor information leak through the masking device.

Bibliography

[1] Dawn Xiaodong Song, David Wagner, and Xuqing Tian. Timing Analysis of Keystrokes and Timing Attacks on SSH. In *USENIX Security Symposium*, 2001.

[2] T. Scott Saponas, Jonathan Lester, Carl Hartung, Sameer Agarwal, and Tadayoshi Kohno. Devices that tell on you: Privacy trends in consumer ubiquitous computing. In *Usenix Security*, volume 3, page 3, 2007.

[3] Charles V. Wright, Lucas Ballard, Scott E. Coull, Fabian Monrose, and Gerald M. Masson. Spot Me if You Can: Uncovering Spoken Phrases in Encrypted VoIP Conversations. In *IEEE Symposium on Security and Privacy*, pages 35–49, 2008.

[4] Qixiang Sun, Daniel R. Simon, Yi-Min Wang, Wilf Russell, Venkata N. Padmanabhan, and Lili Qiu. Statistical Identification of Encrypted Web Browsing Traffic. In *IEEE Symposium on Security and Privacy*, pages 19–30, 2002.

[5] Jean-François Raymond. Traffic Analysis: Protocols, Attacks, Design Issues, and Open Problems. In *Workshop on Design Issues in Anonymity and Unobservability*, pages 10–29, 2000.

[6] Arthur C. Callado, Carlos Alberto Kamienski, Geza Szabo, Balázs Péter Gero, Judith Kelner, Stenio F. L. Fernandes, and Djamel Fawzi Hadj Sadok. A Survey on Internet Traffic Identification. pages 37–52, 2009.

[7] `http://bro-ids.org/;http://www.snort.org`.

[8] Anthony McGregor, Mark Hall, Perry Lorier, and James Brunskill. Flow Clustering Using Machine Learning Techniques. In *PAM*, pages 205–214, 2004.

[9] Andrew W. Moore and Denis Zuev. Internet traffic classification using bayesian analysis techniques. In *SIGMETRICS*, pages 50–60, 2005.

[10] Thomas Karagiannis, Konstantina Papagiannaki, and Michalis Faloutsos. BLINC: multilevel traffic classification in the dark. In *SIGCOMM*, pages 229–240, 2005.

[11] Manuel Crotti, Maurizio Dusi, Francesco Gringoli, and Luca Salgarelli. Traffic classification through simple statistical fingerprinting. pages 5–16, 2007.

[12] Sebastian Zander, Thuy T. T. Nguyen, and Grenville J. Armitage. Automated Traffic Classification and Application Identification using Machine Learning. In *LCN*, pages 250–257, 2005.

[13] Laurent Bernaille, Renata Teixeira, and Kavé Salamatian. Early application identification. In *CoNEXT*, page 6, 2006.

[14] Charles V. Wright, Fabian Monrose, and Gerald M. Masson. On Inferring Application Protocol Behaviors in Encrypted Network Traffic. pages 2745–2769, 2006.

[15] Riyad Alshammari and A. Nur Zincir-Heywood. A flow based approach for SSH traffic detection. In *SMC*, pages 296–301, 2007.

[16] Maurizio Dusi, Alice Este, Francesco Gringoli, and Luca Salgarelli. Using GMM and SVM-Based Techniques for the Classification of SSH-Encrypted Traffic. In *ICC*, pages 1–6, 2009.

[17] Heyning Cheng and Ron Avnur. Traffic analysis of ssl encrypted web browsing. *URL citeseer. ist. psu. edu/656522. html*, 1998.

[18] George Danezis. Traffic Analysis of the HTTP Protocol over TLS. *unpublished paper*, 2009.

[19] George Dean Bissias, Marc Liberatore, David Jensen, and Brian Neil Levine. Privacy Vulnerabilities in Encrypted HTTP Streams. In *Privacy Enhancing Technologies*, pages 1–11, 2005.

[20] Andrew Hintz. Fingerprinting Websites Using Traffic Analysis. In *Privacy Enhancing Technologies*, pages 171–178, 2002.

[21] Marc Liberatore and Brian Neil Levine. Inferring the source of encrypted HTTP connections. In *ACM Conference on Computer and Communications Security*, pages 255–263, 2006.

[22] Charles V. Wright, Lucas Ballard, Fabian Monrose, and Gerald M. Masson. Language Identification of Encrypted VoIP Traffic: Alejandra y Roberto or Alice and Bob? In *In Proceedings of the 16th Annual USENIX Security Symposium*, Boston, MA, August 2007.

[23] Andrew M. White, Austin R. Matthews, Kevin Z. Snow, and Fabian Monrose. Phonotactic Reconstruction of Encrypted VoIP Conversations: Hookt on Fon-iks. In *IEEE Symposium on Security and Privacy*, pages 3–18, 2011.

[24] Tatu Ylönen and Chris Lonvick. The Secure Shell (SSH) Transport Layer Protocol., 2006.

[25] Csaba Kiraly, Giuseppe Bianchi, Fabrizio Formisano, Simone Teofili, and Renato Lo Cigno. Traffic masking in IPsec: architecture and implementation. In *Mobile and Wireless Communications Summit*, pages 1–5, 2007.

[26] Csaba Kiraly, Simone Teofili, Renato Lo Cigno, Matteo Nardelli, and Emanuele Delzeri. Traffic Flow Confidentiality in IPsec: Protocol and Implementation. In *The Future of Identity in the Information Society.*

[27] George Danezis, Claudia Díaz, Carmela Troncoso, and Ben Laurie. Drac: An Architecture for Anonymous Low-Volume Communications. In *Privacy Enhancing Technologies*, pages 202–219, 2010.

[28] Charles V. Wright, Scott E. Coull, and Fabian Monrose. Traffic Morphing: An Efficient Defense Against Statistical Traffic Analysis. In *NDSS*, 2009.

[29] Amir Houmansadr, Giang T. K. Nguyen, Matthew Caesar, and Nikita Borisov. Cirripede: circumvention infrastructure using router redirection with plausible deniability. In *ACM Conference on Computer and Communications Security*, pages 187–200, 2011.

[30] Kevin P. Dyer, Scott E. Coull, Thomas Ristenpart, and Thomas Shrimpton. Peek-a-Boo, I Still See You: Why Efficient Traffic Analysis Countermeasures Fail. In *IEEE Symposium on Security and Privacy*, pages 332–346, 2012.

[31] Xiapu Luo, Peng Zhou, Edmond W. W. Chan, Wenke Lee, Rocky K. C. Chang, and Roberto Perdisci. HTTPOS: Sealing Information Leaks with Browser-side Obfuscation of Encrypted Flows. In *NDSS*, 2011.

[32] Shui Yu, Theerasak Thapngam, Hou In Tse, and Jilong Wang. Anonymous web browsing through predicted pages. In *IEEE Globecom Workshops*, pages 1581–1585, 2010.

[33] Shui Yu, Theerasak Thapngam, Su Wei, and Wanlei Zhou. Efficient Web Browsing with Perfect Anonymity Using Page Prefetching. In *ICA3PP (1)*, pages 1–12, 2010.

[34] Shui Yu, Guofeng Zhao, Wanchun Dou, and Simon James. Predicted Packet Padding for Anonymous Web Browsing Against Traffic Analysis Attacks. pages 1381–1393, 2012.

[35] Fan Zhang, Wenbo He, and Xue Liu. Defending Against Traffic Analysis in Wireless Networks through Traffic Reshaping. In *ICDCS*, pages 593–602, 2011.

[36] Hans-Otto Georgii. *Stochastics: Introduction to Probability and Statistics (de Gruyter Textbook)*. Walter de Gruyter, 1 edition, February 2008. ISBN 3110191458.

[37] `http://www.openssh.com`.

[38] Weka 3: Data Mining Software in Java. `http://www.cs.waikato.ac.nz/ml/weka/`.

[39] Alfonso Iacovazzi and Andrea Baiocchi. Optimum packet length masking. In *International Teletraffic Congress*, pages 1–8, 2010.

[40] Frank Lauren Hitchcock. The distribution of a product from several sources to numerous localities. *Journal of Math. Phys.*, 20:224–230, 1941.

[41] Parv Venkitasubramaniam, Ting He, Lang Tong, and Stephen B. Wicker. Toward an analytical approach to anonymous wireless networking. *Communications Magazine, IEEE*, 46(2):140–146, 2008.