9-12-2014

Neyman-Pearson Decision in Traffic Analysis

Juan Antonio Elices Crespo

Juan Antonio Elices Crespo

Candidate

Electrical and Computer Engineering

Department

This dissertation is approved, and it is acceptable in quality and form for publication:

Approved by the Dissertation Committee:

Gregory L. Heileman , Chairperson

Fernando Perez-Gonzalez

Manel Martinez-Ramon

Jedidiah R. Crandall

Carmela Troncoso

Neyman-Pearson Decision in Traffic Analysis

by

Juan A. Elices

B.S., Computer Science, U.N.E.D., 2008
M.S., Telecommunication Engineering, Universidad Valladolid, 2009
M.S., Electrical Engineering, University of New Mexico, 2011
M.S., Computer Science, U.N.E.D., 2012

DISSERTATION

Submitted in Partial Fulfillment of the

Requirements for the Degree of

Doctor of Philosophy
Engineering

The University of New Mexico

Albuquerque, New Mexico

July, 2014

Dedication

To my parents and to my sister Rosa, that have been the greatest support on all the decisions I have made.

To all my friends who have shared with me the good and bad moments of being so far from home.

Acknowledgments

I would like to express my deepest gratitude to my advisor, Professor Fernando Pérez González, for all his support and guidelines during this period of time. Without him this work would have never been possible. I also express my warmest gratitude to Professor Gregory L. Heileman, for all his help in solving any problems that have raised during this time.

I also owe a great gratitude to Dr. Carmela Troncoso, for her helpful advices on my research during the time I spent in University of Vigo.

I am indebted to the rest of committee members: Professor Jedidiah R. Crandall and Professor Manel Martínez-Ramón.

Also, I would like to thank all my labmates that I have had in the Informatics Group of University of New Mexico, as well as in the Signal Processing in Communications Group of University of Vigo.

Finally, I would like to thank my parents for all the support and love in every occasion. I also want to thank: Mario, Alejandro, Fernando , Iban, Alvaro Diaz, Pancho, LuisFe, Lauren Casey, Fran, Fernando Serrano and a big list of friends that have made me feel in Albuquerque as home. I want to thank specially to Kaile Liang, one of the hardest working persons I have ever met and that has been a model for me in many occasions.

Neyman-Pearson Decision in Traffic Analysis

by

Juan A. Elices

B.S., Computer Science, U.N.E.D., 2008

M.S., Telecommunication Engineering, Universidad Valladolid, 2009

M.S., Electrical Engineering, University of New Mexico, 2011

M.S., Computer Science, U.N.E.D., 2012

Ph.D., Engineering, University of New Mexico, 2014

Abstract

The increase of encrypted traffic on the Internet may become a problem for network-security applications such as intrusion-detection systems or interfere with forensic investigations. This fact has increased the awareness for traffic analysis, i.e., inferring information from communication patterns instead of its content.

Deciding correctly that a known network flow is either the same or part of an observed one can be extremely useful for several network-security applications such as intrusion detection and tracing anonymous connections. In many cases, the flows of interest are relayed through many nodes that reencrypt the flow, making traffic analysis the only possible solution. There exist two well-known techniques to solve this problem: passive traffic analysis and flow watermarking. The former is undetectable but in general has a much worse performance than watermarking, whereas the latter can be detected and modified in such a way that the watermark is destroyed.

In the first part of this dissertation we design techniques where the traffic analyst (TA) is one end of an anonymous communication and wants to deanonymize the other host, under this premise that the arrival time of the TA's packets/requests can be predicted with high confidence. This, together with the use of an optimal detector, based on Neyman-Pearson lemma, allow the TA deanonymize the other host with high confidence even with short flows. We start by studying the forensic problem of leaving identifiable traces on the log of a Tor's hidden service, in this case the used predictor comes in the HTTP header. Afterwards, we propose two different methods for locating Tor hidden services, the first one is based on the arrival time of the request cell and the second one uses the number of cells in certain time intervals. In both of these methods, the predictor is based on the round-trip time and in some cases in the position inside its burst, hence this method does not need the TA to have access to the decrypted flow.

The second part of this dissertation deals with scenarios where an accurate predictor is not feasible for the TA. This traffic analysis technique is based on correlating the inter-packet delays (IPDs) using a Neyman-Pearson detector. Our method can be used as a passive analysis or as a watermarking technique. This algorithm is first made robust against adversary models that add chaff traffic, split the flows or add random delays. Afterwards, we study this scenario from a game-theoretic point of view, analyzing two different games: the first deals with the identification of independent flows, while the second one decides whether a flow has been watermarked/fingerprinted or not.

Contents

List of Figures

List of Tables

Glossary

AS	Autonomous System
AUC	Area Under the ROC Curve
cdf	Cumulative Distribution Function
CIDR	Classless Inter-Domain Routing
DSSS	Direct Sequence Spread Spectrum
EFF	Electronic Frontier Foundation
HHM	Hidden Markov Model
HTTP	Hypertext Transfer Protocol
HTTPS	Hypertext Transfer Protocol Secure
i.i.d	Independent and Identically Distributed
IBW	Interval Based Watermark
ICBW	Interval-Centroid-Based Watermark
IPD	Inter-Packet Delay
IP	Internet Protocol
ISP	Internet Service Provider

JSD	Jensen-Shannon Divergence
KDE	Kernel Density Estimation
KLD	KullbackLeibler Divergence
MLE	Maximum Likelihood Estimation
MLP	Multi-layer Perceptron
NAT	Network Address Translation
NSA	National Security Agency
NE	Nash Equilibrium
NTP	Network Time Protocol
P2P	Peer-to-peer
PDV	Packet Delay Variation
PMF	Probability Mass Function
PDF	Probability Distribution Function
RMSE	Root-Mean-Square Error
ROC	Receiver Operating Characteristic
RTT	Round-trip Time
TA	Traffic Analyst
TCP	Transmission Control Protocol
TLS/SSL	Transport Layer Security / Secure Sockets Layer
XOR	Exclusive OR.

SPE	Subgame Perfect Equilibrium
SSH	Secure Shell
SVM	Support Vector Machine
XOR	Exclusive OR

Chapter 1

Introduction

In modern society, the use of communication networks is increasing dramatically, especially the Internet. Although it has been really well accepted and it is difficult to imagine today's world without this technology, its use entails some risks. Very important information flows through the network that a malicious agent can eavesdrop.

This problem has made Internet users become more concerned about the privacy and security of their information using encrypted channels to transmit it. For instance, many applications, such as web servers, are migrating to be accessed using TSL/SSL protocols to assure the confidentiality and integrity of the transmitted data. This concern is contributing to the massive growth of encrypted traffic through the Internet.

However, with the additional security of encrypted channels come potential tradeoffs, such as the lack of visibility into encrypted traffic that can hide potential threats to a firewall or an intrusion prevention system. This fact has increased the awareness for traffic analysis, i.e., inferring information from communication patterns instead of its content. In fact, traffic analysis can be performed even when the messages are encrypted and cannot be decrypted.

Using the detailed profile of an observed communication, traffic analysis tries to reveal important information, such as the identities of the parties communicating, their locations and even the content of the communication. As most technologies, traffic analysis can be used for "good" purposes, such as security enforcement, forensics and investigation, or "evil" purposes, such as gathering private information or to penetrate particular security protocols and thus violate traditional security properties.

1.1 Traffic analysis

1.1.1 Origins in the military

Traffic analysis has its origins in the military context, where the most common scenario is radio traffic analysis. This is the study of intercepted or monitored wireless traffic for the purpose of gathering intelligence information without recourse to cryptanalysis [1].

Information Sources

The communication elements that are subject to study by radio traffic analysts are [2, 3]:

- Callsigns: series of letters and/or numbers assigned that identify a specific radio station.
- Frequencies: each organization using radio communications is allotted various blocks of the radio frequency spectrum.
- Schedules: Military radio station networks usually have a prearranged schedule for contacting other stations and sending messages. Discovering these schedules

is an important task of traffic analysis to allocate its monitoring resources most efficiently, also, it might be possible to use communications schedules to identify stations.

- Address systems: radio stations often used message address systems to route messages to particular addressees or military units, several of which might be served by a single radio station.

- Operator chatter: idle chatter between radio operators, with no valuable information, can be unencrypted. Chatter collected from careless radio operators can disclose information such as callsigns, frequencies, or contact schedule information.

- Transmission patterns: some operators had distinctive transmission patterns. For example, ground forces would sign off in a different way than air forces.

- Location of the source: finding the azimuth (line bearing between the source of the signal and the receiving station) of a propagated radio signal from different locations allows to identify the position of the transmitter.

Counter-measures

Measures taken to avoid traffic analysis include:

- Changing frequencies, callsigns, and communications transmission schedules according to a predetermined plan. Usually frequency rotations and change callsigns are done simultaneously in order to difficult a listening attempt.

- Reducing the communication time.

- Encrypt addresses and operator chatter.

- Send dummy traffic to mislead the traffic analyst.

- Sending a continuous encrypted signal, whether or not traffic is being transmitted. This is also called masking or link encryption.

History

Figure 1.1: Beeston Hill 'Y' station in Sheringham, Norfolk, UK. [4]

The first documented use of traffic analysis dates from 1904 when the Japanese Navy during the the Russo-Japanese War located a squadron of the Czar's Vladivostok fleet which was cruising secretly south of Tokyo Bay by intercepting a message [5].

But it was during the World War I when traffic analysis became an indispensable information source. A well-known example is the Y stations (Figure 1.1) deployed by the British to gather radio information that were used in both world wars.

During World War II, traffic analysis had a significant role in the most relevant battles:

- Battle of the Atlantic: the combined use of traffic analysis, radio-direction finding and cryptanalysis (ULTRA) helped to locate the U-boat wolfpacks [6].
- Battle of the Coral Sea, where the presence of Japanese in the Solomon Islands was revealed [7, p. 93].

- Battle of Midway, the advanced warning that the entire Japanese combined fleet was en route allowed to plan American strategy in advance [7, p. 144, 155, 165, 166].

- Battle of Stalingrad: the Germans took appropriate defenses when the Soviet unit was moving towards Stalingrad as they intercepted a radio communication [8, p. 203].

Also countermeasures to traffic-analysis had a great impact to confuse the enemy:

- Pearl Harbour, where the Japanese aircraft carrier radio were transferred ashore on November 15, 1941, and the radio operators kept generating dummy packets with a common pattern [9, p. 140-142].

- Operation Quicksilver in Normandy which had the purpose of simulating an invasion through Pas-de-Calais instead of Normandy. A main part of this operation was creating wireless traffic simulating both the apparent movement of a whole army to Southeastern England and the embarkation preparation that this invasion would need [10, p. 127-128].

If the reader is interested in deepening in the field of history of traffic analysis and its use during military operations, we recommend the following texts:

- The Origination and Evolution of Radio Traffic Analysis [11, 12]

- The history of traffic analysis: World War I - Vietnam [2]

- A Century of Spies: Intelligence in the Twentieth Century [13]

- Chicksands, a Millennium of History [14]

- Intelligence Power in Peace and War [15]

1.2 Traffic Analysis in the Internet

Traffic analysis techniques can be applied to Internet communications in such a way that private and sensitive information can be extracted from encrypted communications.

In contrast to military powers, the adversary is many times assumed to have limited resources and not be able to intercept all the communications. Examples could be commercial entities, local governments, law enforcement, criminal organizations or terrorist networks, even this assumption may not hold for some governments or law enforcement agencies, such as the National Security Agency (NSA) which has a capacity to eavesdrop a large proportion of Internet traffic. In fact, the Wall Street Journal estimated it in 75% of all Internet traffic of the U.S. [16].

In the following, we illustrate the most relevant previous work using traffic analysis.

Traffic Analysis of Secure Shell (SSH)

SSH is a cryptographic network protocol for secure data communication that allows users to remote login and execute commands. SSH's encryption is intended to provide confidentiality and integrity of data over an unsecure network. However, as SSH only transmits a packet per key stroke, it leaks a lot of information.

Song et al. [17] showed that using a hidden Markov model (HMM) they were able to reduce the entropy of a password by 1.2 bits per character pair when the passwords are generated from a uniform distribution. This reduction is achieved when the attacker does not have typing statistics from the victim, but, of course, knowing the user's typing profile improves performance.

Monrose et al. [18] showed that passwords can be hardened using keystroke dy-

namics as every user's typing pattern varies. This property can be used also by an eavesdropper to identify the user that is connecting to an SSH server, particularly after a long sequence has been observed.

Side-Channel leaks to obtain keyboard strokes

Measuring the inter-packet delays (IPDs) in SSH communications is not the only way of figuring out the keys that a user is pressing. Obtaining the keystrokes constitutes a serious information leakage as an attacker could easily obtain the passwords a user is typing.

Asonov and Agrawal [19] and Zhuang et al. [20] showed that with acoustic emanations it is possible to infer the pressed keys. Also, the electromagnetic emanations can be used to recover the typed keys as shown by Vuagnoux and Pasini [21].

Zhang and Wang [22] demonstrated that in a multi-user environment, another user can learn the instants another user is pressing keys from the stack information disclosed by its virtual file within `procfs`. Afterwards, they use a HMM to obtain the sequence of pressed characters.

Marquardt et al. [23] showed that a phone placed by a keyboard can be used to recover text entered on the keyboard, using the vibrations information captured by the accelerometers.

Cai and Chen [24] showed that on mobile phones, it is feasible that one application recovers the keys typed on a different one, reading the motion and vibrations of the screen.

Ristenpart et al. [25] demonstrated that an Amazon Elastic Computing Cloud (EC2) user can intentionally place a virtual machine on the same physical machine as another users virtual machine, which allows the former to estimate the cache usage, traffic load and keystroke timing.

Web Fingerprinting

A web page fingerprinting attack is an attempt to identify the web page a user is visiting using an encrypted connection. This attack can be done in IPSec tunnels, SSH tunnels, and on anonymity networks such as Tor. The attacker first builds profiles of web-site pages, and afterwards, its goal is to infer which web page is being retrieved.

Mistry and Raman [26] and Cheng and Avnur [27] were among the first to demonstrate the feasibility of inferring the objects downloaded via encrypted Secure Sockets Layer (SSL) connections using the transmitted data volumes.

Hintz [28] and Sun et al. [29] proposed two different methods of web fingerprinting. While Hintz demonstrates his attack only for a small number of websites using SafeWeb, Sun et al. presented a larger validation.

Sun et al.'s classifier is based on the Jaccard's coefficient obtaining a 75% correct identification. They also propose some countermeasures that include: padding, mimicking another web page or morphing the traffic. These attacks need to be able to differentiate connections so they are effective to SSL connections but not under tunneling protocols.

The first attempts to fingerprint that were effective on tunneling protocols are Bissias et al. [30] and Liberatore and Levine [31]. On the first one, the detector is the correlation coefficient between the packet size, and on the second one, there are two possible detectors: a) naive Bayes, and b) Jaccard's coefficient, in both the fingerprints are only based on packet lengths, discarding timing information.

Herrmann et al. [32] proposed a detector using a multinomial-naive Bayes classifier that outperforms the previous models. The authors claimed that they could identify up to 97% of the requests on SSH and virtual private network (VPN) tunnels, obtaining low accuracy on low-latency anonymity systems as Tor [33] and Java

Anonymous Proxy (JAP) [34] due to equal size fragments.

Panchenko et al. [35] used a support vector machine (SVM) on some features based on volume, time, and direction, with this detector they improved the accuracy in Tor from 3% to 55% and in JAP from 20% to 80% in a close-world scenario and in an open-world scenario they achieved a 73% correct detection for a 5% of false positive probability, they proposed to obfuscate the traffic by loading several pages simultaneously.

Coull et al. [36] proved that it is feasible to identify target web pages within anonymized NetFlow data using kernel density estimation for mapping anonymized addresses to logical servers and afterwards check constraints using a binary Bayes belief network.

Chen et al. [37] have further showed that web fingerprinting cannot only infer the web page that is being accessed, but also sensitive information, such as medical records and financial data.

Web Privacy

A remote web server accessed by a client can infer information about the web-sites that the client has previously browsed [38]. The malicious web-server can determine this information by measuring the time the user's browser requires to load an object from other web-sites. Since browsers perform various forms of caching, the time required for this action depends on the user's browsing history.

Jackson et al. [39] proposed two additional ways. The first is based on reading the color of the displayed links, and the second on a cooperative site tracking where several domains agree to share the user's visited history on their domains, for which they redirect the user to a central tracking authority. They claimed that the only way to prevent this attack is to disable the features of the browser that use persistent-

state.

Jang et al. [40] developed a framework for JavaScript to detect privacy-attacks on information flows, showing that several popular sites perform this kind of attacks.

Machine identification

Kohno et al. [41] proposed how to determine if two different IP addresses are indeed the same device. The method is based on the clock skew (the amount by which the clock drifts per unit of time), that can identify a particular machine, as it is different for all machines even for identical models from the same manufacturer. Therefore, if the clock drift of two remote machines seems to match for a long time, they conclude that the targets are indeed the same machine. The technique can be used even when the machines use Network Time Protocol (NTP) for clock synchronization.

Murdoch [42] used a similar approximation to detect hidden servers in anonymous networks such as Tor [33]. They modulate the number of requests send to this server. These requests make the CPU load increase, heating up the machine with the inevitable result of modifying the clock skew. If the provided time of this machine can be measured, and the corresponding pattern of clock skew changes, it is possible to link the machine with its hidden identity. Zander and Murdoch [43] modified this scheme sampling at the desired instants in order to reduce the quantization error.

The opposite problem has also been widely considered, i.e., identify the machines behind a certain IP address. This problem is relevant due to the common use of network address translator (NAT) gateways and firewalls, that make all the connections from a private LAN to appear with the same source IP. Bellovin [44] addressed this problem by noticing that on many operating systems, the IP headers ID field is a simple counter. However, this technique is becoming less effective because many operating systems have started to randomize this field to prevent 'idle scanning'. Beverly [45] showed that the TCP/IP stack varies with the operating system (OS)

and its version, which allows the attacker to identify the number of machines behind a NAT gateway. This OS fingerprint can also be used for an attacker to identify possible vulnerabilities. A similar approach was used by Franklin et al. [46] to identify wireless device drivers.

Detecting stepping stones

A stepping stone is a compromised host that is being used as an attack platform, relaying traffic to another remote destination for the real attacker to hide their identity. Detecting stepping stones involves a firewall that sees incoming and outgoing connections, and tries to determine if a pair of them may be carrying the same stream. If so, it might mean that the a host is compromised and used as a stepping stone.

This problem is one of the main applications of the research presented in this dissertation, and in Chapter 2 we deal with the different proposed strategies used on the previous literature.

Location privacy

Location privacy is the ability of an individual to move in public space with the expectation that under normal circumstances their location will not be systematically and secretly recorded for later use.

Nowadays, location information can be gathered silently, and cheaply. That allows studying individual movements of individuals and making models to predict somebody's position. This can be a serious concern not only because of the loss of privacy but also due to safety reasons.

Brockman et al. [47] showed that human traveling behaviour can be modeled as a continuous time random walk. Humans follow simple reproducible patterns in most of their travels as shown in [48] and these movements can be explained by

periodical movements or social relationships[49] that allow the attacker to predict future movements. This predictability is potentially very high and could achieve a 93% of correct prediction [50]. An interesting fact discovered by De-Montjoye et al. [51] is that four spatio-temporal points are enough to uniquely identify 95% of the individuals.

Data can be collected from the wireless access points the user connects to [52, 53], cell phone base station [48, 54, 50], location-based Social Networks [55, 56, 57, 58] or GPS position [58, 59].

It can be concluded that using location data, sensitive information about one's identity, relations and intentions can be inferred merely though traffic analysis.

1.3 Anonymous communications: Avoiding traffic analysis in Internet

Privacy preserving communications, that do not leak any residual information from their metadata, have been studied since Chaum [60] introduced the "mix". Since then, research in anonymous communications has been extended to many areas.

1.3.1 Mixnet-based schemes

The mixnet family of protocols use a chain of proxies or mix servers, each one receives messages from multiple senders, shuffles them, and sends them back out in random order to the next destination (see Figure 1.2).

Each message is encrypted in layers in such a way that each proxy can decrypt its own layer of encryption to reveal where to send the message next. Hence, mixes only know the node that they immediately received the message from, and the immediate

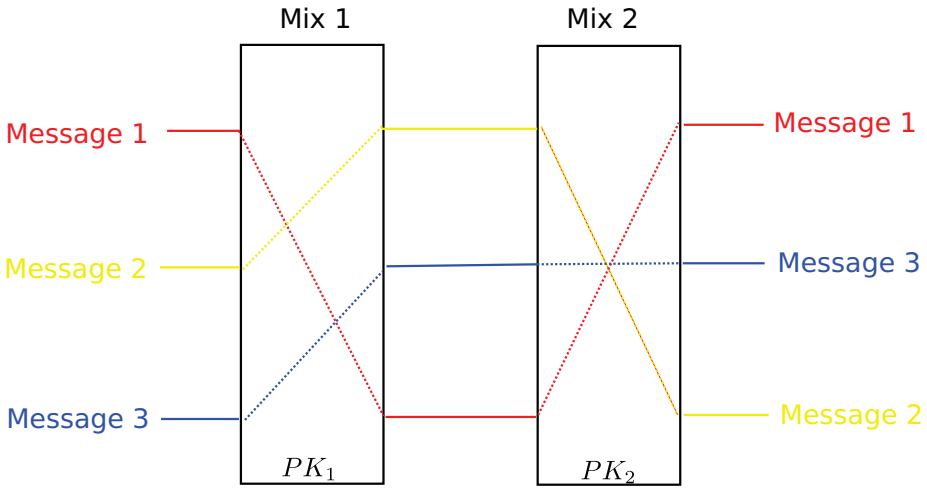


Figure 1.2: Mixnet network.

destination to send, making the network resistant to malicious mix nodes.

Anonymous remailers

Cypherpunk remailers or type I remailers [61] were the first widespread public implementation of mixnet. They provide an anonymous store and forward architecture for sending emails.

The next generation of remailers, type II, was introduced by Cottrell [62]. Type II remailers are more resistant to traffic analysis as they send messages in fixed-size packets, preventing attacks based on correlating messages by size. The reusable reply blocks that they implement pose a security risk because they allow multiple messages to be sent.

Mixminion [63] (type III remailer) avoids the reply problem by making reply messages indistinguishable by a mix node to forward messages. Messages are transferred between remailers using a Transport Layer Security (TLS) protected tunnel with ephemeral keys so material gathered by a passive adversary is useless after the ephemeral keys are deleted.

Onion routing

Onion routing [64, 65, 66, 67] is the equivalent of mix networks, but in the context of circuit-based routing. Instead of routing each anonymous packet separately, the first packet opens a circuit. Then, the rest of packets are routed through this same circuit. Finally, a message to close the path is sent.

The circuit is opened sending a message encrypted in layers, that can only be decrypted by a chain of onion routers using their respective private keys. This message contains the addressing information about the next node, as well as material shared between the sender and the routers. It has become clear that in the absence of heavy amounts of cover traffic, patterns of traffic are present that could allow an attacker to follow a stream in the network and identify the communicating parties [68]. This kind of attacks will be addressed later in this dissertation.

Onion routing aims at providing anonymity in low-latency applications where mix networks cannot be used because of their excessive delay, and in order to be fast enough to be useful, its mixing strategy has to be minimal. This has resulted in successful timing-correlation attacks.

Zero Knowledge designed the Freedom network following the architecture of onion routing [69, 70] but it did not succeed on achieving enough widespread acceptance to cover its operating costs.

Tor: the second-generation onion router

Tor was developed in 2004 by the Onion Router project team as the second generation of the onion router [33].

Tor relays Internet traffic through free, worldwide volunteer servers called onion routers or relays. Onion routers communicate with one another, and with users'

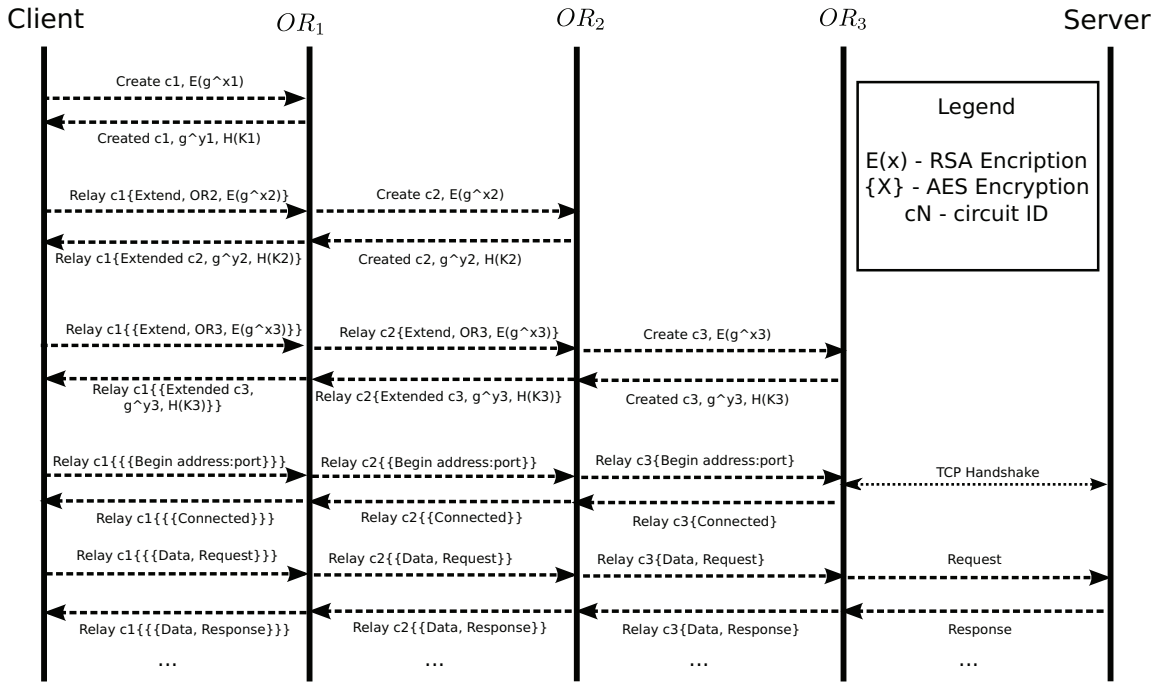


Figure 1.3: Tor circuit creation and data transmission

onion proxies (Tor clients), via TLS connections.

Prior to sending data through Tor, a client has to establish a circuit. For this purpose, the onion proxy chooses three Tor relays over which their communication is relayed. Instead of a single message to create the whole circuit, the onion proxy constructs circuits incrementally, negotiating a symmetric ephemeral key (session key) with each onion router on the circuit, one hop at a time. This guarantees forward secrecy and compulsion resistance: only short term encryption keys are ever needed.

Once the onion proxy has established the circuit (so it shares keys with each relay on the circuit), it can send relay cells that contain application data. When an onion router receives a relay cell, it decrypts the cell using the session key for that circuit, and if valid, the onion router sends the cell with one less encrypted layer to the next hop. On the reply channel, each relay adds an encryption layer using the single key

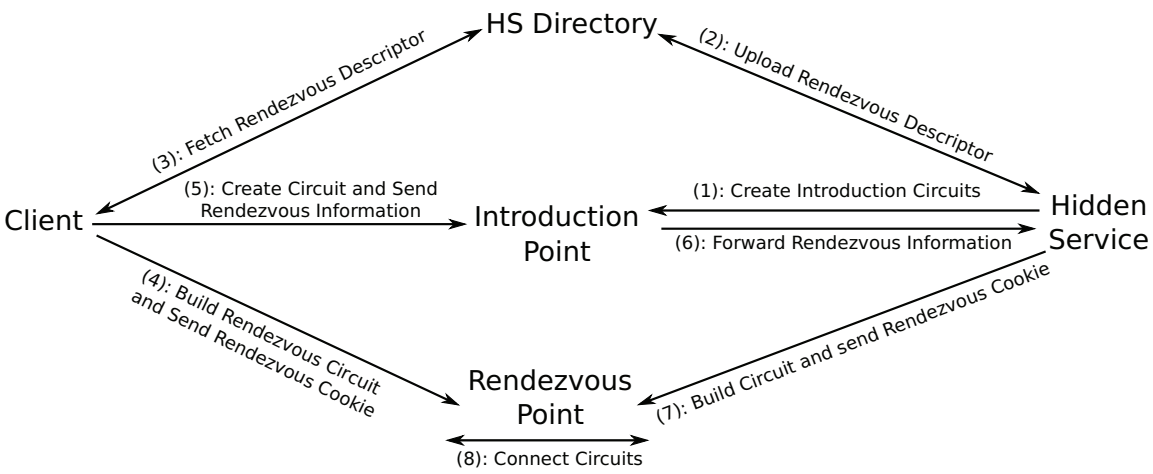


Figure 1.4: Establishing and accessing a hidden service

of the circuit, sending the cell back toward the onion proxy along the circuit. This whole process is depicted in Figure 1.3, where we represent a layer of encryption using {}.

Tor also provides responder anonymity with location-hidden services, that allows to offer TCP services without revealing their IP address.

The process to establish and access a hidden service is shown in Figure 1.4. First, the hidden server creates introduction circuits with several Tor relays called introduction points (step 1). Then the hidden service uploads its descriptor that includes which onion routers are its introduction points to the hidden service directory (step 2). When a client wants to access this hidden service, the client fetches the hidden service descriptor (step 3), afterwards the client chooses one Tor relay as rendezvous point creating a circuit towards it, next the client sends an arbitrary 20-byte value (rendezvous cookie) through this circuit (step 4). The client also builds a separate circuit to one of the hidden services introduction points, through it the client sends the rendezvous point and the cookie (step 5), that is forwarded to the hidden service (step 6). Finally, the hidden service creates a circuit to the rendezvous point sending the rendezvous cookie through it (step 7) so the rendezvous point can join both

circuits together (step 8).

In this way, the client and the hidden server does not know the other party of the communication.

Web MIXes

Web MIXes [71] were designed for anonymous and unobservable real-time Internet access that can prevent traffic analysis and flooding attacks. This scheme has been implemented as a web-anonymizing proxy called JAP.

The design goal is to secure anonymous communication, even in the presence of a very powerful adversary, that can observe all communications on the network (global passive), modify the communications on the links by delaying, injecting or deleting messages, and control all but one of the mixes.

Web MIXes use a cascade of mixes, where each message is processed by all mixes in the same order. After the cascade of mixes, the cache-proxy sends the data to the Internet and receives answers from the servers. To prevent traffic correlation attacks, Web MIXes send the same amount of data per slot. Unfortunately, while this idea is practical in the context of circuit-switching telecommunication networks, in modern IP networks is very expensive to guarantee a constant bandwidth.

1.3.2 DC-net systems

Dining Criptographers problem

This problem was proposed by Chaum [72], as an illustrative example to show how it was possible to send anonymous messages with unconditional sender and recipient untraceability.

The problem is stated as follows: three cryptographers gather around a table for dinner, where the waiter informs them that the meal has been paid by someone, who could be one of the cryptographers or the NSA. The cryptographers respect each other's right to make an anonymous payment but want to find out whether the NSA paid.

The protocol is performed in two stages. In the first stage, every two cryptographers establish a shared one-bit secret, i.e. cryptographer A and B share a secret bit , A and C share another, and B and C share another. In the second stage, each cryptographer publicly announces a bit, the XOR of the two shared bits they hold with their two neighbours if they did not pay the meal and the opposite bit if they did pay the meal. A XOR of all the announced bits determines whether the NSA or any of the cryptographers paid, but the identity of who paid remains unknown to the other cryptographers.

Under the assumption of a reliable broadcast network, the anonymity of the senders is proved to be unconditional, without relying on a trusted third party and even if the attacker is computationally unbounded. The problems of this method are that only one participant is allowed to transmit at a time using this protocol, and that a malicious player can jam the protocol so that the final XOR result is useless. This attack cannot be detected easily since each node is as anonymous as any initiator.

To solve this problem, Chaum proposed to detect dishonest players via a system of traps. To jam the DC-net a dishonest player must transmit a message in a slot that was not assigned to him, but if the slot in which he tries to transmit is a trap, then the attacker may be detected during a decommitment phase.

Herbivore [73] is a peer-to-peer scalable anonymous communication system based on DC-nets, where the network is divided into smaller anonymizing cliques of k-nodes within which a DC-net protocol is implemented.

Dissident [74] is a messaging protocol that offers provable anonymity with accountability for moderate-size groups. To this end, it shuffles a $N \times N$ matrix of pseudorandom seeds, then uses these seeds in N "pre-planned" DC-nets protocol runs.

1.3.3 Network routing-based techniques

CROWDS

Crowds [75] aims to provide an anonymous access to web servers. Each user contacts a central server and receives the list of relays. Then, a user relays her web requests by passing them to another randomly selected node in the crowd. This node can either submit the request directly to the end server or forward it to another randomly chosen member, and in the latter case this process can be repeated by the next node. The random choice is biased in favor of forwarding to a new node. The anonymity of initiators is based on the assumption that the members in the crowd cannot know if the previous node was the actual requester or was just passing the request along.

However, it has been demonstrated that Crowds is vulnerable to predecessor attacks [76]. This is because, if a node repeatedly requests a particular resource, the node and the resource can eventually be linked.

Buses for anonymous delivery

The metaphor of messages being like the public transport system, where each piece of information is allocated a seat within the bus was presented in [77].

Messages (buses) traverse the network using deterministic routes, since the messages traverse the network in fixed routes, the adversary cannot learn whether there is any communication between the nodes or not.

Each time the bus gets to a node, this node checks which messages were for it and ignores those containing dummy information. Then it changes the messages that correspond to its seats to either a message or a dummy.

1.3.4 Peer to peer anonymous communications systems

Tarzan

Tarzan is a peer-to-peer anonymous IP network overlay [78] that achieves its anonymity with layered encryption and multihop routing, like a Chaumian mix.

A message initiator chooses a path of peers pseudo-randomly through a restricted topology. Tarzan uses cover traffic to prevent a global observer from using traffic analysis to identify an initiator.

Tarzan uses a restricted network topology for packet routing. Each node maintains persistent connections with a small set of other nodes, this set is called a mimic. Routes of anonymous messages are created only through mimics and between mimics to avoid traffic analysis through links with insufficient traffic. Unfortunately, a security weakness of Tarzan is that the selection of neighboring nodes for the mimic's structure is done on the basis of a network identifier or address, which is vulnerable to spoofs and also due to the small subset of known nodes a malicious node on the path can identify uniquely the originator with very high probability as it knows three nodes of the subset [79]. A new version of Tarzan forcing each node to know all other nodes, which is clearly less practical.

MorphMix

MorphMix [80] has a similar architecture and threat model to Tarzan. An important difference is that the route is chosen by intermediate nodes, observed by witnesses

and trusted by the user. MorphMix includes a collusion detection mechanism that monitors for any cliques in the selection of nodes in the path, but this mechanism has also been compromised [81].

1.4 Contributions and Publications

In the following, a brief summary of the contributions contained in this thesis together with the published works that support their research value is provided.

1.4.1 Publications

The publications that support the research undertaken during the period of this thesis are the following:

Journal Papers

J1 J. A. Elices and F. Pérez-González, "A highly optimized flow-correlation attack," *Elsevier Computer Networks*, 2014, submitted.

Conference papers

C1 J. A. Elices, F. Pérez-González, and C. Troncoso, "Fingerprinting Tor's hidden service log files using a timing channel," in *Information Forensics and Security (WIFS), 2011 IEEE International Workshop on*, 29 2011-dec. 2 2011, pp. 1 –6.

C2 J. A. Elices and F. Pérez-González, "Fingerprinting a flow of messages to an anonymous server," in *Information Forensics and Security (WIFS), 2012 IEEE International Workshop on*. IEEE, 2012, pp. 97–102.

C3 J. A. Elices and F. Pérez-González, "Locating tor hidden services through an interval-based traffic-correlation attack," in *Communications and Network Security (CNS), 2013 First IEEE Conference on*, 2013.

C4 J. A. Elices and F. Pérez-González, "The Flow Fingerprinting Game," in *Information Forensics and Security (WIFS), 2013 IEEE International Workshop on*. IEEE, 2013.

C5 J. A. Elices and F. Pérez-González, "Locating Tor hidden services through an interval-based traffic analysis," in *Communications and Network Security (CNS), 2014 Second IEEE Conference on*, 2014 (submitted).

Unpublished

U1 J. A. Elices and F. Pérez-González, "Linking correlated network flows through packet timing: a game-theoretic approach," 2013.

1.4.2 Contributions

The main contributions that have been made during the research detailed in this thesis can be summarized in the following points. We indicate between brackets the produced publications related to each of them:

- Proposed three different methods to detect correlated flows, i.e. flows that carry the same stream. The first one is based on predicting the time a packet should appear on the correlated flow ([C1], [C2]), the second on the IPDs ([J1, C3, U1]) and the third on the number of packets that fall inside certain intervals ([C4], [C5]).
- We proposed a game-theoretic framework to study the impact of an active attacker to the correlation ([C3, U1]) in the second method.

- We studied the common application of flow correlation: stepping stones detection and compromising anonymity([J1,C3,U1]). But we have also proposed how to detect if a flow has a certain flow embedded. This is used for locating hidden servers of anonymous networks.

1.5 Organization of the rest of the dissertation

After having put a contextual framework in this chapter describing traffic analysis and anonymous communications, the remaining chapters of the thesis are structured as follows:

- Chapter 2 formally defines the problem we want to solve, describe the applications we want to address and shows the state-of-the-art.

- Chapter 3 describes a prediction-based method for leaving traces in log files.

- Chapter 4 describes a prediction-based method for correlating flows. We apply this method to deanonymize Tor hidden services.

- Chapter 5 proposes a method for correlating flow based on detecting an increment of packets that fall in the intervals. We also apply it for deanonymizing Tor hidden services.

- Chapter 6 describes a method using the IPDs, which, therefore, does not need any delay predictor. We apply this method to detect stepping stones and to correlate flows that go through Tor.

- Chapter 7 studies the scenario when an active-adversary is present in the method given in 6 using a game-theoretic approach.

- Chapter 8 provides the conclusions that can be drawn from the research developed in this thesis, pointing out the future research lines that this thesis opens.

Chapter 2

Problem Description and Previous Approaches

2.1 Notation

In this section we introduce the notation that we use in the sequel.

Random variables are denoted by capital letters (e.g., X), and their individual realizations by lower case letters (e.g., x). The domains over which random variables are defined are denoted by script letters (e.g., $\mathcal{X}$). Sequences of n random variables are denoted by X^n if they have random nature or by x^n if they are a particular realization. X_i or x_i indicate the ith element of X^n or x^n, respectively. The pdf (probability density function) of a random variable X is denoted by $f_X(x)$, $x \in \mathcal{X}$. We use the same notation to refer to the joint pdf of a sequence, i.e. $f_{X^n}(x^n)$, $x^n \in \mathcal{X}^n$. When no confusion is possible, we drop the subscript in order to simplify the notation. We denote by Δ the difference operation of a sequence, i.e $\Delta x^n = \{x_2 - x_1, \ldots, x_n - x_{n-1}\}$ and by $\Delta_i x^n = x_{i+1} - x_i$ the ith element of this sequence.

The indicator function of x being in the interval $[a, b]$ is denoted by

$$\mathbb{1}_{[a,b]}(x) = \begin{cases} 1 & \text{if } x \in [a, b], \\ 0 & \text{if } x \notin [a, b]. \end{cases} \tag{2.1}$$

The Kullback-Leibler Divergence (KLD), that is a non-symmetric measure of the difference between two probability distributions f_P and f_Q, is defined as follows:

$$D_{KL}(P, Q) = \sum_{i \in \mathcal{P}} f_P(i) \log \left(\frac{f_P(i)}{f_Q(i)} \right) \tag{2.2}$$

The Jensen-Shannon divergence (JSD), D_{JS} [82], is a metric for two probability distributions f_P, f_Q based on the Kullback-Leibler divergence (KLD) as follows:

$$D_{JS}(P, Q) = \sqrt{\frac{1}{2} \left(D_{KL}(P \mid\mid M) + D_{KL}(Q \mid\mid M) \right)} \tag{2.3}$$

where $M = \frac{1}{2}(P + Q)$ is the mid-point measure.

2.1.1 Performance Metrics

To measure performance, we use two metrics: the probability of detection (P_D) and the probability of false positive (P_F). Given two hypotheses: H_0 and H_1, P_D is the probability of deciding H_1 when H_1 holds, whereas P_F is the probability of deciding H_1 when H_0 holds. Sometimes we use the probability of misdetection defined as $1 - P_D$.

Typically, performance is graphically represented using the so-called Receiver Operating Characteristic (ROC) curves, which represent P_D vs. P_F. In order to compare different ROCs in a simple way, we use the Area Under the ROC Curve (AUC), a measure that takes a value of 1 in the case of perfect detection and 0.5 in the case of random choice.

2.1.2 Neyman-Pearson Lemma

The Neyman-Pearson lemma [83] is a statistical theorem that guarantees that when performing a hypothesis test between two hypotheses H_0 and H_1 the likelihood ratio test is the most powerful test of size α. This means that given a fixed probability of false positive, i.e., $P_F = \alpha$, the likelihood test gives the highest true positive rate (probability of detection).

The likelihood test rejects H_0 in favor of H_1 when:

$$\Lambda(x) = \frac{L(H_1|x)}{L(H_0|x)} \leq \eta, \text{ where } P(\Lambda(x) \leq \eta|H_0) = \alpha, \tag{2.4}$$

this means that η is a threshold to obtain $P_F = \alpha$.

2.2 Brief Introduction to Game Theory

Game theory is the mathematical study of interaction among intelligent rational decision-makers. Formally, a two player game is defined as a quadruple $G(A_1, A_2, u_1, u_2)$, where $A_i = \{a_{i,1}, \dots a_{i,n_i}\}$ are the actions available to the i player, $u_i : A_1 \times A_2 \mapsto \mathbb{R}$, $i = 1, 2$ is the utility function or payoff of the game for player i. An action profile is the double $a \in A_1 \times A_2$. We are interested in zero-sum games, where $u_1(a) + u_2(a) = 0, \forall a \in A_1 \times A_2$, which means that the gain (or loss) of utility of player 1 is exactly balanced by the losses (or gains) of the utility of player 2. In this case, we can simplify the game notation to a triplet $G(A_1, A_2, u)$, where $u = u_1 = -u_2$.

We say that an action profile $(a_{1,i^*}; a_{2,j^*})$ represents a Nash equilibrium (NE) if

$$\begin{aligned} u(a_{1,i^*}; a_{2,j^*}) &\geq u(a_{1,i}; a_{2,j^*}) \ \ \forall a_{1,i} \in A_1 \\ u(a_{1,i^*}; a_{2,j^*}) &\leq u(a_{1,i^*}; a_{2,j}) \ \ \forall a_{2,j} \in A_2, \end{aligned} \tag{2.5}$$

intuitively this means that none of the players can improve his utility by modifying his strategy assuming the other player does not change his own.

Games can be classified in simultaneous games, where both players move unaware of the other player action, and sequential games, where later players have some knowledge about earlier actions. In sequential games, an action profile is a subgame perfect equilibrium (SPE) if it represents a NE of every subgame of the original game. Therefore, a SPE is a refinement of the NE that eliminates non-credible threats.

2.3 Problem Description

The problem addressed in this thesis is the identification of correlated flows. We say that two flows are correlated if they follow a common timing pattern due to sharing the same source (i.e. the unencrypted payload of part of the packets is the same). This means that:

- Packets can be added or/and removed at an encryption layer.
- Packets can be added to the flow.
- Several flows can merge into one and the traffic analyst (TA) cannot separate them.
- Packets can be removed from the flow.

There are two general approaches for finding correlated flows: passive analysis and active watermarks. Passive analysis schemes are based on correlating some characteristics of the flows, such as packet timings or packet counts, without altering such flows [84, 85, 86]. On the other hand, active watermarks actively modify the flow by delaying individual packets to embed a watermark signal. This watermark

Figure 2.1: Flow correlation problem

signal can be embedded in individual delays between packets [87, 88] or in some properties of the intervals [89, 90, 91].

We also define flow fingerprinting, which is a flow watermarking scheme with a unique modification to each flow, so that every source sequence can be indistinctively identified. Note that the terms non-blind watermark and fingerprint are many times interchangeable.

We outline the problem of finding correlated flows in Figure 2.1. The goal of the TA is to accept or reject the hypothesis that a flow y^{n_2} consisting of n_2 packets is indeed the same flow as a known one, x^n, of length n packets. In the case we are in an active watermark scenario, the TA can modify the flow by embedding a known signal or watermark w^n. Due to the nature of the problem the modification must be additive, i.e., $x^n = u^n + w^n$, where $w_i \geq 0,\ i = 0, \ldots, n$. In the case that we deal with bidirectional flows, we add a direction subscript: I for the flow from TA creator to TA detector, and B for the opposite flow. In chapter 3 instead of X and Y representing a packet flow they denote an HTTP request flow, that has only the I direction.

We denote by D^n the network delay of each packet, hence

$$Y_{x_i} = x_i \pm D_i,\ i = 1, ..., n, \tag{2.6}$$

where Y_{x_i} represents the component of Y^{n_2} that corresponds to x_i, and the sign varies depending on the direction, plus for I and minus for B.

Let us define the hypotheses:

$$H_0 : y^{n_2} \text{ is not correlated with } x^n$$
$$H_1 : y^{n_2} \text{ is correlated with } x^n.$$

In Chapters 3, 4 and 5 time is divided into intervals. We define the ith interval or period for the TA creator as

$$P_{i,c} = (b_{i,c}, e_{i,c}) = (b_{i,c}, b_{i,c} + T_{i,c}), \quad i = 0, \ldots, L-1, \tag{2.7}$$

where $b_{i,c}$ is the beginning of each interval, $e_{i,c}$ is the end, $T_{i,c}$ is the interval length and L is the number of considered intervals. For the TA detector, the expression is identical but changing the subscript c for d. Unless otherwise stated, we assume that intervals do not overlap, i.e.

$$P_{i,c} \cap P_{j,c} = \emptyset, \text{ and } P_{i,d} \cap P_{j,d} = \emptyset, \forall i \neq j. \tag{2.8}$$

When there is no possible confusion about where the interval is measured, we drop the subscript c or d in order to simplify the notation.

We also define the random variables C_i and S_i that count the number of packets that fall inside $P_{i,c}$ and $P_{i,d}$, hence:

$$C_i = \sum_{j=1}^{n} \mathbb{1}_{P_{i,c}}(X_j) \tag{2.9}$$

and

$$S_i = \sum_{j=1}^{n_2} \mathbb{1}_{P_{i,d}}(Y_j). \tag{2.10}$$

The packets at the detector can have either one of two sources: the flow by the TA creator or a different source (chaff or other flows). We represent this as $S^L = E^L + R^L$, where E^L is the number of packets that traverse the TA creator and R^L denotes the remaining packets. Formally,

$$E_i = \sum_{j=1}^{n} \mathbb{1}_{P_{i,d}}(Y_{x_j}). \tag{2.11}$$

2.3.1 Fingerprinting a hidden service log

In Chapter 3, we consider a different application. We assume a scenario where the TA do not have access to the TA detector but he/she wishes to leave a fingerprint in its log so as to use it later as a proof that a particular machine was indeed the receiver of the HTTP flow. This can be used in network forensics to prove that a certain machine once hosed a particular content, even if this content has been permanently deleted. This fingerprinting problem is conceptually identical to the flow-correlation problem.

2.4 Previous Approaches

In this section we explain how this problem, detecting correlated flows, has been addressed previously, dividing these methods in passive analysis and active watermarking according to the non-modification or modification of the flow, respectively. Afterwards, we present some attacks against the invisibility of active watermarking.

2.4.1 Passive Analysis

Passive analysis schemes attempt to detect the correlation of two flows extracting some characteristics of both, such as packet timings or packet counts, without modifying the flows. The decision is based on correlating those extracted characteristics.

Zhang and Paxson

Zang and Paxson [84] proposed to correlate the traffic by measuring the time when both flows are in OFF state (i.e., no transmission for at least T_{idle}).

- They consider that two OFF periods are correlated if their end times differ by less than δ, where δ is a control parameter.

- Two flows F_1 and F_2 have a consistent ordering if once observed that F_1 ends its OFF period before F_2, then F_1 should always ends its OFF period before F_2.

- They consider two connections C_1 and C_2 are a stepping stone connection pair if

$$\frac{OFF_{1,2}}{\min(OFF_1, OFF_2)} \geq \gamma, \;\; OFF^*_{1,2} \geq min_{csc}, \text{ and } \frac{OFF^*_{1,2}}{\min(OFF_1, OFF_2)} \geq \gamma' \tag{2.12}$$

  where OFF_1 and OFF_2 are the number of OFF periods in connections C_1 and C_2, respectively, $OFF_{1,2}$ is the number of correlated OFF periods, $OFF^*_{1,2}$ is the number of consecutive coincidences with consistent order, and min_{csc}, γ and γ' are control parameters.

The method in [84] achieves a large confidence when connections are several minutes long (i.e., thousands of packets) but not so much reliability on short connections. They do not consider any alteration in the traffic.

Donoho, Flesia, Shankar, Paxson, Coit and Staniford

Donoho et al. [85] studied how to detect a stepping stone when it delays packets up to a maximum tolerable delay constraint to evade detection. They conclude that provided that a large enough sequence is available, they can correlate the traffic regardless of the modification. They use wavelets to separate the short-term behavior from the long-term behavior, and use the correlation on the latter.

They analyze Poisson streams, due to their mathematical properties, but claim that the results are generalizable to other distributions. They also claim that in the

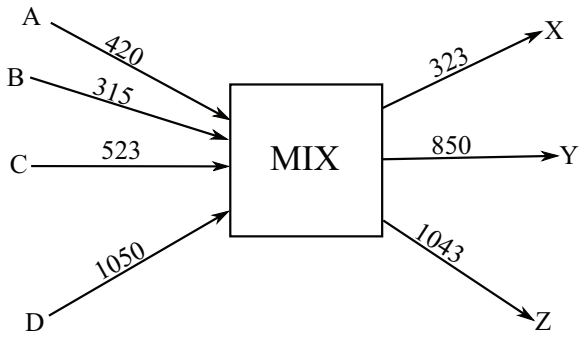


Figure 2.2: Packet counting to a mix network

presence of chaff packets the correlation parameter converges to a value that depends on the chaff traffic but is strictly larger than 0, so they can still detect stepping stones as long as the flow is long enough.

Blum, Song, and Venkataraman

Blum et al. [86] also studied stepping-stone detection under a maximum tolerable delay constraint. They count the difference between the number of packets in both flows. When this difference goes over a certain threshold they conclude that the flows are not correlated.

In the case that the stepping-stone can introduce a certain amount of chaff traffic, the threshold is incremented linearly with it. They also claim that under unlimited chaff traffic, the attacker can always evade detection.

Packet Counting Attack

Serjantov and Sewel [92] proposed a packet count attack. This attack correlates the number of packets that input and output flows carry in a period of time. This attack is effective against low-latency mixes, including Tor onion routers. This is shown on Figure 2.2, where the number on the arrow represents the number of packets observed

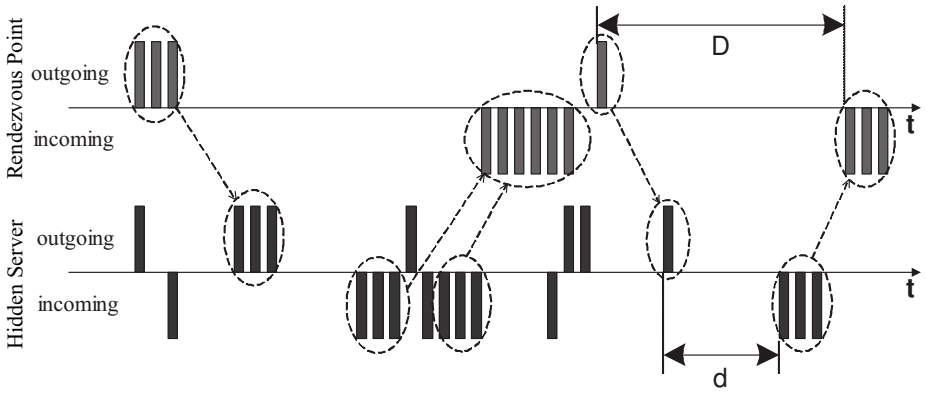


Figure 2.3: Packet Counting Attack using timing information [94]

by an attacker on the link over one time period. In the example, it is plausible that the flow from A and C goes to Y, B's to X and D's to Z.

A very similar approach of counting the number of packets in periods of time is followed by Levine et al. [93]. They propose to drop packets to mitigate the attack.

A modification of this attack is used by Øverlier and Syverson [94] to locate hidden servers when the attacker controls the first onion router in the circuit between the hidden service and the rendezvous point. They propose to use the timing between bursts to correlate the flows, for instance D and d in Figure 2.3. The packets that seem to have no correspondence in the other flow are considered to come from the process of setting up the tunnelled connections and are not considered for the correlation. The considered bursts are marked in Figure 2.3.

2.4.2 Active Watermarking

In this section, we present the state-of-the-art methods to identify correlated flows using active watermarks. Recall that active watermarking is a technique that modifies the flow at the TA creator to incorporate a certain signal or watermark. This signal is embedded by delaying certain packets.

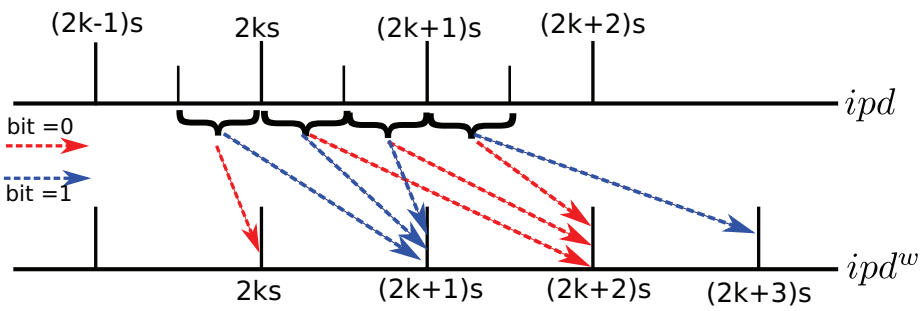


Figure 2.4: IPD modification in quantization watermark

Quantization Watermarking

Wang and Reeves [87] proposed the first active flow watermark. The watermark is embedded in the inter-packet delays (IPDs). They first quantize the IPD and embed one bit of information by either adding half of the quantization step or not, as shown in Figure 2.4.

They argue that with sufficient redundancy the watermark can always be detected even if a timing perturbation is added to each packet. Hence, the drawback of this method is the amount of packets needed to obtain a good performance.

Interval-based watermarking

Pyun et al. [90] proposed an interval-based watermark (IBW) designed to resist attacks that modify the number of packets, such as flow splitting, chaff packets and repacketization.

Starting from a random offset, the unwatermarked flow is divided into intervals I_i of length T_{IBW}. A bit is encoded in the difference of packets between two consecutive intervals, I_k and I_{k+1}. As shown in Figure 2.5, a 0 is embedded by making the number of packets within I_k larger than I_{k+1}, to this end the packets within intervals I_{k-1} and I_{k+1} are delayed by T_{IBW}, so they fall into I_k and I_{k+2}, respectively. To embed

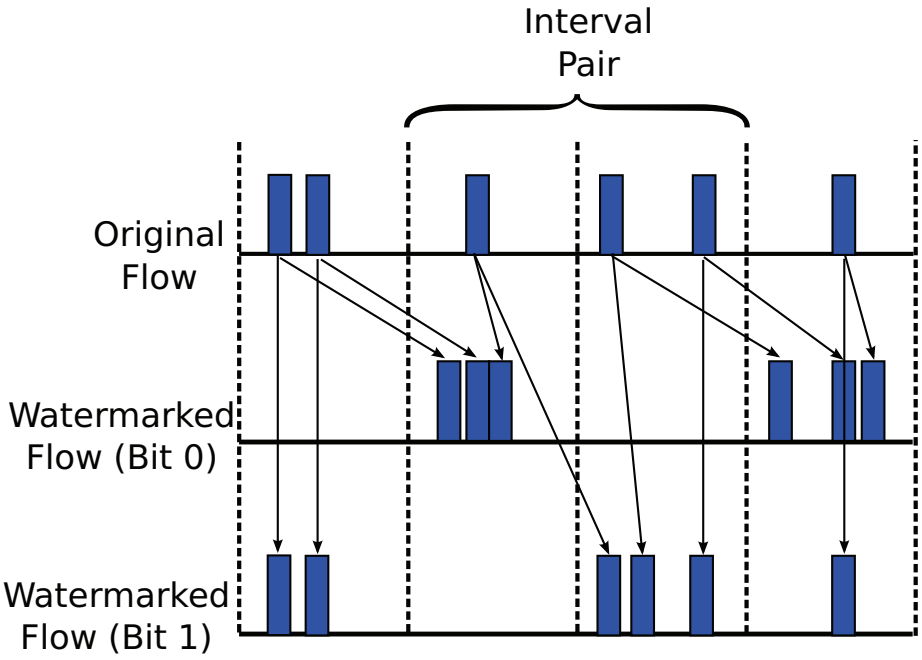


Figure 2.5: Example of Interval-Based Watermark

a 1, IBW tries to make the number of packets in I_{k+1} larger than I_k, for which it delays all the packets in I_k by T_{IBW}, so that they fall into the interval I_{k+1}.

Tracking Anonymous Peer-to-Peer VoIP Calls

Wang et al. [95] proposed a watermark to trace VoIP connections, based on the idea that the IPD of these connections at the source has a very small variance. They embed the watermark in the "pseudo-IPDs" defined as $x_{i+d} - x_i$, i.e., the difference between the timing of two packets separated by $d-1$ other packets.

To embed a bit, they select $2r$ packets where r is the redundancy of a bit. These packets are separated into two groups, G_0 and G_1, and then they delay by a milliseconds all the packets of one of these groups. To decode this bit, they calculate the difference between the mean of the "pseudo-IPDs" of each group, i.e.,

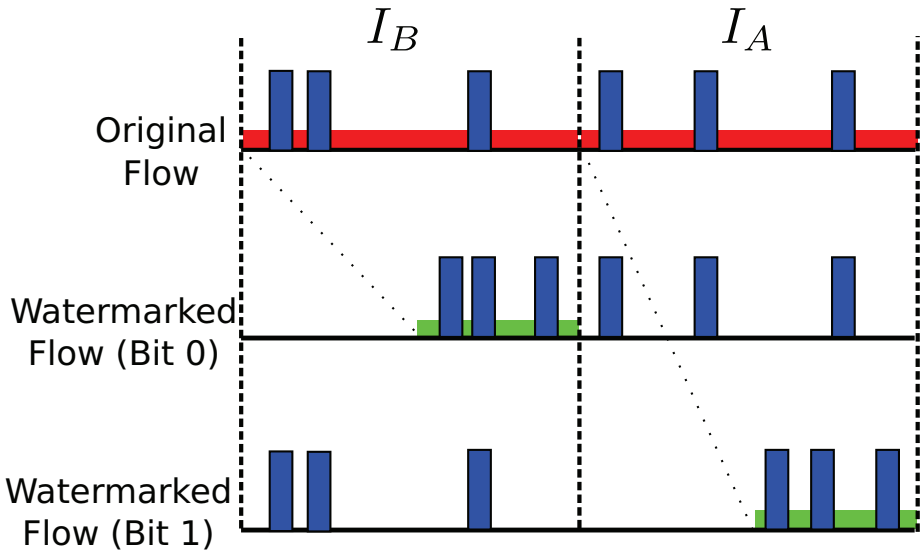


Figure 2.6: Example of Interval-centroid-based Watermark

$\frac{1}{r}(x_{i+d} - x_i - \frac{1}{r}(x_{j+d} - x_j),\ i \in G_1, j \in G_0$, and the decision is based on the sign of this difference.

Interval-centroid-based Watermarking

Wang et al. [91] proposed Interval-centroid-based watermarking (ICBW). As its name suggests, the watermark is encoded by modifying the centroid of certain intervals.

Starting from a random point, the flow to be watermarked is divided into $2n$ intervals of length T_{ICBW}. Then, ICBW randomly partitions the intervals into two groups A and B, each containing n intervals. Afterwards, ICBW randomly determines which intervals of each group are used to embed a particular watermark bit. As shown in Figure 2.6, to encode bit '1' ICBW deliberately increases the centroid of the intervals from group A mapping the whole interval (shown in red) into a smaller

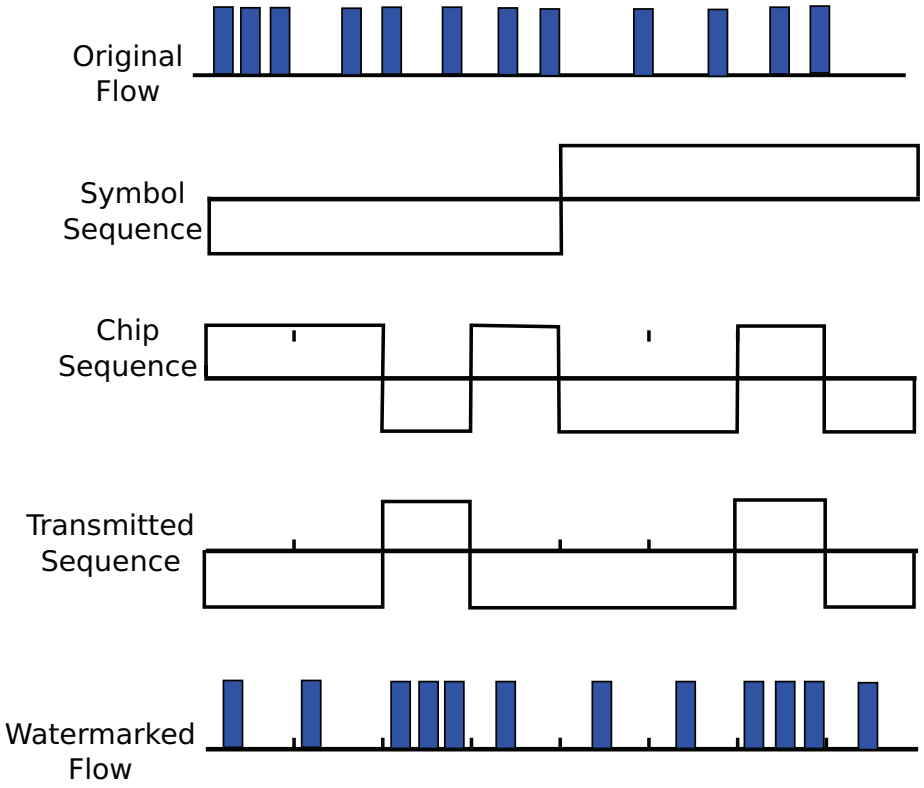


Figure 2.7: Example of DSSS Watermark

subinterval at its end (depicted in green). To encode bit '0', ICBW deliberately increases the centroid of intervals from group B in a similar manner.

Direct-sequence Spread Spectrum Watermarking

Yu et al. [96] proposed an interval watermark based on DSSS (Direct Sequence Spread Spectrum) communication techniques in order to become invisible.

A DSSS signal is created by XORing the pseudonoise sequence and the symbol sequence. This DSSS signal is embedded by modifying the traffic rate. As shown in Figure 2.7, in a flow with average rate of D a 1 is embedded by increasing its rate up to $D + A$ and a 0 by decreasing the rate to $D - A$. This method requires very long sequences.

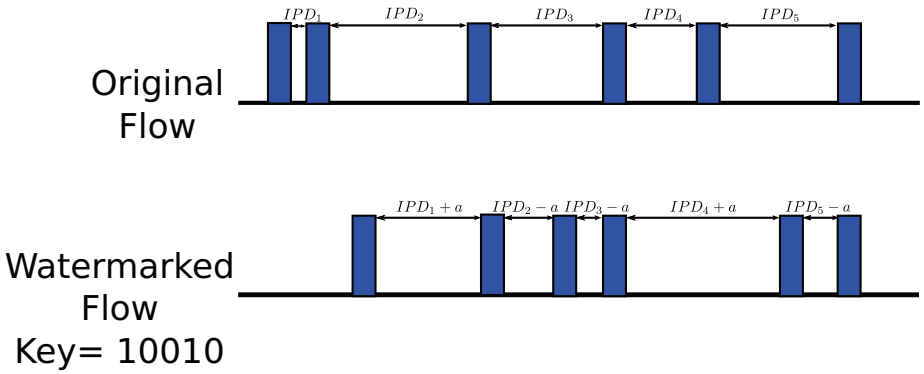


Figure 2.8: Example of RAINBOW Watermark

RAINBOW

Houmansadr et al. [88] proposed RAINBOW (Robust And Invisible Non-Blind Watermark), a non-blind watermark which is robust to packet drops and repacketization.

RAINBOW records the IPDs of the unwatermarked flow, then it embeds the

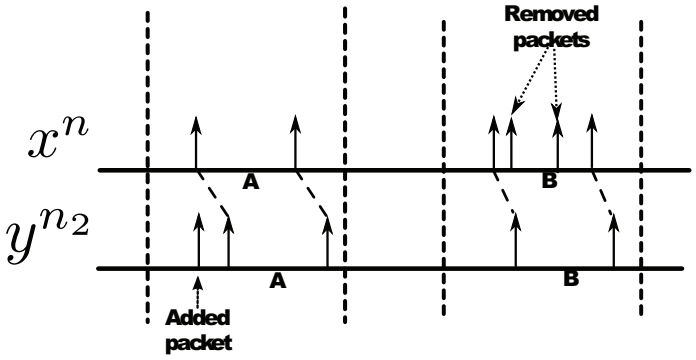


Figure 2.9: Selective Correlation in RAINBOW Watermark [88]

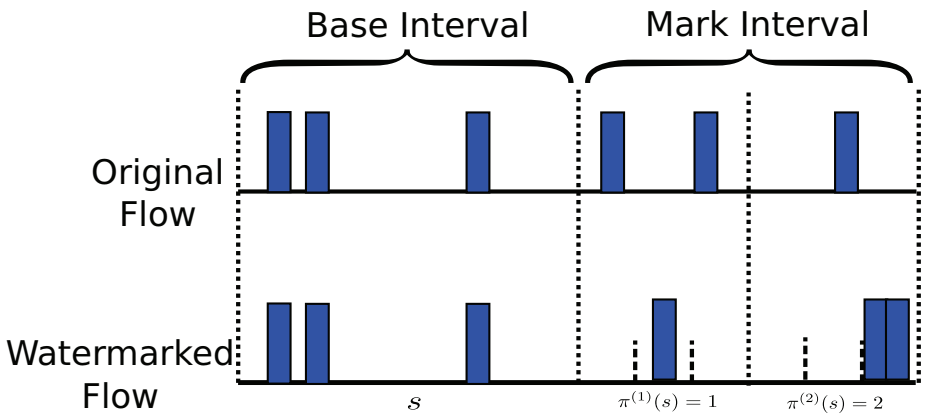


Figure 2.10: Example of SWIRL Watermark

watermark by modifying the IPDs by a different quantity ($+a$ for 1, $-a$ for 0, or vice versa) as shown in Figure 2.8. The detector calculates the normalized correlation between the received IPDs minus the original IPDs and the watermark sequence, if this value goes over a threshold then the detector concludes that the received sequence is watermarked. In order to deal with added and lost packets, it uses "selective correlation" that removes the packets that have no correspondence in the other flow as shown in Figure 2.9.

SWIRL

Houmansadr and Borisov [89] proposed SWIRL (Scalable Watermark that is Invisible and Resilient to packet Losses).

Starting from a random offset α, SWIRL divides a flow into intervals of length T_{SWIRL}: half of them, called base intervals, are used to determine the slots pattern, and the other half, mark intervals, are used to actually embed the watermark. SWIRL associates one base interval to one mark interval conditioned to the fact that the base interval is previous to the mark one.

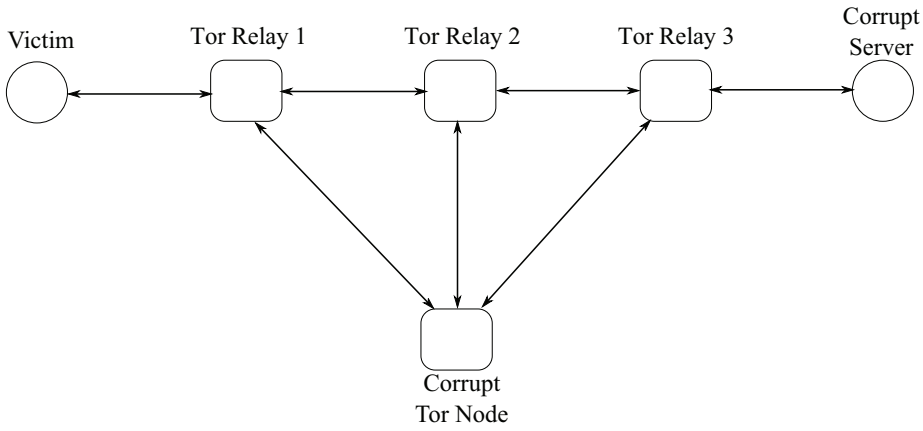


Figure 2.11: Low-Cost Traffic Analysis of anonymous networks

The base intervals are used to calculate a symbol s in the range $[0, m)$, as follows:

$$s = \lfloor qmC/T_{SWIRL} \rfloor \bmod m \tag{2.13}$$

where C is the centroid of the interval and q is a quantization multiplier that helps smooth out the distribution.

The value s is then used to transform the associated mark interval. The mark interval is divided into r subintervals, so that each one is further divided into m slots. SWIRL selects a slot in each subinterval i by applying a permutation π^i to s. Each packet is delayed such that it falls within a selected slot, possibly moving it into the next subinterval. This process is shown in Figure 2.10.

Low-Cost Traffic Analysis of anonymous networks

Murdoch and Danezis [97] proposed an attack against onion routing networks in which a malicious server sends the data modulated in a very specific traffic pattern. The goal of the attacker is to identify which relays this flow is traversing.

To this end, the malicious server needs to control only one corrupt relay that can create connections to other relays. By measuring the latency of these connections,

the attacker can identify that the modulated traffic is going through the other relay. The whole process is depicted in Figure 2.11.

The attacker modulates traffic to be very distinguishable, sending bursts of traffic (sending as fast as Tor allows) of 10 to 25 seconds followed by a silence for a period between 30 to 75 seconds. The problem of this method is that a very recognizable pattern needs to be used for being detected, but this also makes the attack easily noticeable to the anonymous network or the client.

Cell counter watermarking for Tor Connections

Ling et al. [98] proposed to embed a watermark in the number of cells sent together, i.e., the size of a burst. They apply this watermark against the Tor network.

They embed a '1' by sending a burst of 3 cells and a '0' by sending a single cell alone. Although a burst of cells can be combined or divided in the intermediate node, the authors claim that these modifications can be predicted and reversed in most of the cases.

2.4.3 Detecting Watermarks

There is an inherent problem in watermarking schemes. They can be detected and this allows an attacker, e.g. the stepping stone, anonymous network, etc., to easily modify the timing of the watermarked flow in such a way that the watermark is removed.

Peng, Ning and Reeves

Peng et al. [99] showed how the quantization watermark can be detected and replicated. They use a sequential likelihood ratio test to differentiate between the hy-

potheses that the delays come from a normal distribution (flow is not watermarked) or that the delays come from a sum of a normal plus a uniform (flow is watermarked).

Afterwards, they recover the parameters of the watermarking algorithm: the quantization step and the proportion parameter, using Expectation Maximization. This method is an iterative optimization method to find the maximum likelihood estimate of parameters that can deal with some missing data.

Results show that they can always detect the presence of the watermark and in most of the cases (around 95%), they can obtain the parameters correctly with the possibility of replicating the watermark in a different flow.

Multi-flow Attacks Against Network Flow Watermarking Schemes

Kiyavash et al. [100] discovered that by overlapping several network flows watermarked using the same key, it is feasible not only to detect the watermark but also to extract the parameters and the key. This is due to the fact that these flows share a common pattern created by the watermark and from this pattern the parameters can be obtained.

This technique can be used to detect any watermark for which the watermarked pattern is not a function of the input, concretely IBW, ICBW and DSSS watermarks are proven to be susceptible to this attack.

BACKLIT

Luo et al [101] showed that any practical timing-based traffic watermark causes noticeable alterations in the intrinsic timing features typical of TCP flows, and so it can be easily detected. Specifically, they use three features to detect anomalies:

- Request-response time (RRT)

- IPDs
- Burst size

They classify the traffic into two groups:

- Bulk transfer traffic: the number of response packets is much larger than the number of request packets, e.g., HTTP.
- Interactive traffic: the number of request packets and response packets are similar, e.g., SSH.

When dealing with interactive traffic, the watermark is detected by anomalies in the RRTs. RAINBOW and SWIRL are detected on the basis that the difference between RRTs is much sparser than the common unwatermarked flows, and IBW and ICBW are easily detected by looking at the proportion of packets with very large RRTs, this is due to the fact that these watermark schemes delay packets by a large quantity (hundreds of milliseconds).

In the case of burst traffic, the watermarks are detected using the distribution of the IPDs. IBW and ICBW are detected because the IPDs larger than the RTT (IPDs between bursts) are not concentrated around the RTT as it would be expected in an unwatermarked flow. RAINBOW and SWIRL are detected using the IPDs smaller than the RTT (IPDs inside the same burst) as they are not concentrated around 0. They also show that the rate of one-packet bursts can be used to detect watermarks in burst traffic.

Lin and Hopper

Lin and Hopper [102] also studied how to detect watermarks, concentrating their efforts in detecting the "invisible" RAINBOW and SWIRL watermarks. First, they define two classes of adversaries:

- Isolated Adversary: the adversary has to decide if a flow is watermarked or not just given this flow.
- Chosen-flow adversary: the adversary has to decide if a received flow is watermarked or not, controlling the timing information of that flow at its source.

RAINBOW is detected by a chosen-flow adversary seeing the distribution of the jitter, specifically the percentage of samples in a neighborhood of 0 ms. In the case of an isolated adversary, they use the cosine similarity between the cumulative distribution function (cdf) of the received flow IPDs and the cdf of unwatermarked IPDs of similar flows.

SWIRL is detected by a chosen-flow adversary sending packets evenly separated, this way the quantized centroid will not change. Hence, the marked flow will have the same pattern of cleared and occupied intervals. In the case of an isolated adversary, the attacker uses a naive clustering algorithm that groups packets by their arrival times, then the attacker uses the maximum time-span among all the clusters to determine if the flow is watermarked or not.

2.5 Discussion

As the previous section illustrates, we have seen a proliferation of new watermarking schemes that claim to be invisible followed by attacks against their invisibility. The watermarks are often designed to avoid a specific attack, for instance SWIRL and RAINBOW were designed with the aim of avoiding the multi-flow attack, but afterwards new attacks against its invisibility have been proposed.

In light of these facts, one may ask whether it is possible to construct a perfectly invisible watermark. Unfortunately, this is not possible, as the distribution of the delays is changed. It is not even possible to create a perfectly invisible watermark

against an attacker that uses only first order statistics. Formally, a perfect invisible watermark against this attacker would require $D_{KL}(D||D+W) = 0$ [103]. Due to the fact that the domain of W is $\mathbb{R}^+$, i.e., we cannot add a negative delay, the only possible distribution for perfect invisibility is $f_W(w) = \delta(w)$, i.e., no watermark embedded.

In the case that the distribution of D is unknown for the attacker, and only the distribution of ΔD is available for him, a perfect invisible watermark against this attacker requires $D_{KL}(\Delta D||\Delta D + \Delta W) = 0$. In this case, the construction of a perfect invisible watermark against a test based on first order statistics of the PDV is feasible, but watermark amplitudes are very limited.

Another generalized problem we see is that all these designs are not constructed over a mathematical model that guarantees optimality in any sense. For instance, at the TA creator, one would be interested in finding the watermark that gives the best performance under certain constraints, or at the TA detector, knowing which is the best possible test for detecting the presence of the watermark.

These two observations motivate this dissertation. First, our traffic analysis methods are designed so they can be implemented in a passive analysis situation, and second, we use a mathematical framework that guarantees optimality under some assumed conditions. Specifically, we use detectors based on Neyman-Pearson lemma, explained in Section 2.1.2, and in Chapter 7 we use a game-theoretic framework to design an optimal fingerprint and an optimal detector in the presence of an adversary who tries to impair the correlation.

Chapter 3

Fingerprinting Log Files

3.1 Introduction

This chapter deals with the application presented in Section 2.3.1, i.e., fingerprinting a log file. We consider a scenario where we want to leave a fingerprint on a Tor's hidden server log with the purpose of proving when the machine is confiscated that this particular machine actually hosted the hidden service during some time by finding this fingerprint in the log.

Detecting such a pattern is not straightforward, mainly because of two issues. First, Tor's high delay variability that makes hard to predict the time logged by the hidden service. Second, when detecting a fingerprint it is not possible to distinguish between the our log entries and the ones resulting from other users' requests. We overcome these problems by estimating the log time from the date field included in the HTTP responses to our requests, and by statistically modelling the number of other users' entries in the log file. Our fingerprinting algorithm can be tuned to achieve a probability of misdetection as small as desired, while minimizing the probability of false positives.

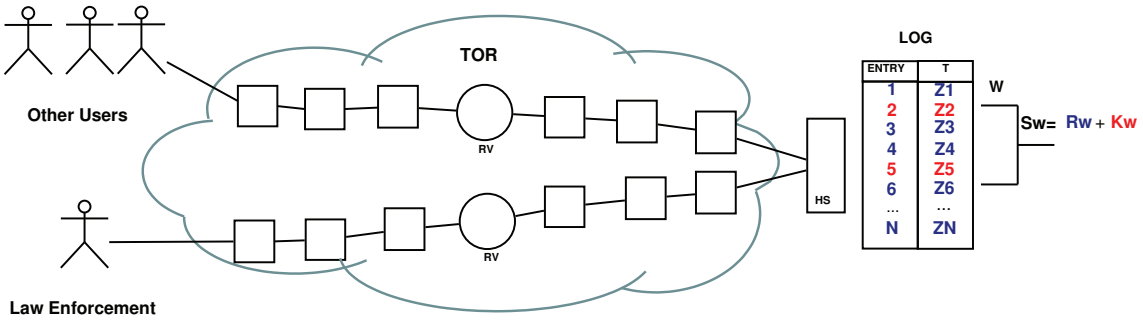


Figure 3.1: System Model

3.2 Formal Problem Description

In this section, we explain the problem and recall the notation that we defined in Chapter 2 as well as that one specific to this chapter.

Figure 3.1 shows the basic model of our problem. We send n HTTP requests to a hidden server through Tor. These requests will appear in the server's log file mixed with other clients' requests (represented in blue and red, respectively, in Figure 3.1). Note that in order to achieve our low detectability goal we do not tag the requests in any way, we just use the timing information.

We consider that the ith request we send appears on the hidden service's log at time

$$Y_{x_i} = x_i + D_i,\ i = 1, ..., n \tag{3.1}$$

recall that D_i is random delay introduced by the Tor network. Loesing et al. modeled D_i as a Fréchet distribution [104], but we choose not to use this approximation since an estimator $\hat{Y}_{x_i}$ of the log time is available in the HTTP response, as we will discuss later.

We denote by F^l the sequence of number of log entries per unit of time (granularity of the log), where l is the number of units of time available in the log. These entries can correspond to our fingerprinting requests that we denote as E^l, or to

other clients' requests, whose number is denoted by R^l.

The sequence R^l is modeled in Section 3.4 as a negative binomial distribution.

Formally, we can express this problem via classical hypothesis testing with the following hypotheses:

H_0: The log has not been fingerprinted.

H_1: The log has been fingerprinted.

In a practical setting, one fixes a certain value of P_F (that has to be very small if we want to achieve a high reliability and avoid accusing an innocent server) and then measure P_D (which we would like to be as large as possible).

In addition, we want to avoid that the presence of the fingerprint can be easily detected by any other party than the originator of such sequence, i.e., the sequence should have low detectability.

3.3 HTTP Response Date Information

In this section we characterize the estimator of the log time, $\hat{Y}_{x_i}$. On the header of the HTTP response, there is the "date field", which we use as an estimator of Y_{x_i}. According to [105], in theory, this field represents the moment just before the HTTP response is generated. We define the estimation error as $\varepsilon_i \doteq Y_{x_i} - \hat{Y}_{x_i}$.

In order to characterize ε, we performed some experiments on an AMD Turion 64 X2 2GHz with 3GB of memory running Apache 2.2.15 over Windows Vista SP2. The experiments were selected to cover a wide range of server situations:

1. Normal situation: We request a 44 bytes file 2 times per second.

2. Large transmission time: We request a 7 MBytes file every minute.

3. Very loaded server: We request a 7 MBytes file 10 times per second.

4. Demanding requests: We request a dynamic file of around 80 bytes, for which the server needs at least 5 seconds to generate the response.

	Code 200 with $\varepsilon = 0$	Code 200 with $\varepsilon \neq 0$	Code 5xx logged	Code 5xx not logged
Experiment 1	94.10%	0.00%	0.01%	5.89%
Experiment 2	93.60%	0.00%	0.91%	5.49%
Experiment 3	11.33%	0.00%	33.78%	54.90%
Experiment 4	100.00%	0.00%	0.00%	0.00%

Table 3.1: Percentage of responses in Apache 2.2.15

Table 3.1 shows the percentage of HTTP responses that we receive with a given response code. We can see that for valid HTTP responses (i.e., the ones with response code 200) the estimation error is $\varepsilon = 0$.

The above results were obtained using only one machine with one server software. In order to assess that those results can be generalized we performed a second experiment. We requested one petition per second during 24 hours to a web server at the University of Vigo. This machine is an AMD Athlon XP 2000+ with 512MB of memory that runs Apache 1.3.33 over Debian 4.0. Table 3.2 shows the obtained pmf (probability mass function) of ε, and Figure 3.2 plots the cdf of the same distribution.

An important conclusion is that ε depends on the machine and its running HTTP server software. In this work we assume that the data from Table 3.2 (also depicted in Figure 3.2) can be extrapolated. We also assume that no HTTP requests triggering an error appear in the log.

$\varepsilon(s)$	$\boldsymbol{Pr(\varepsilon)}$	$\varepsilon(s)$	$\boldsymbol{Pr(\varepsilon)}$	$\varepsilon(s)$	$\boldsymbol{Pr(\varepsilon)}$
<0	0.0000%	7	0.0011%	15	0.0000%
0	67.1689%	8	0.0022%	16	0.0000%
1	31.7958%	9	0.0011%	17	0.0000%
2	0.7881%	10	0.0110%	18	0.0000%
3	0.0773%	11	0.0066%	19	0.0155%
4	0.1214%	12	0.0011%	20	0.0000%
5	0.0055%	13	0.0000%	≥ 20	0.0143%
6	0.0055%	14	0.0000%		

Table 3.2: Probability mass function of ε in Apache 1.3.33

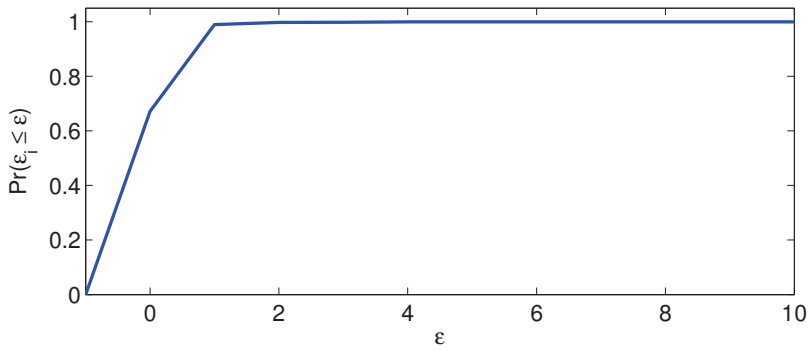


Figure 3.2: Cumulative Distribution Function of ε in Apache 1.3.33

3.4 Modeling the number of Log Entries

In the previous section we explained how to deal with the first noise source, i.e., the HTTP server's delay in logging the request. In this section, we present a model for the number R^l of requests from other clients.

3.4.1 Data Collection

The access logs for this research were obtained from seven different World Wide Web servers: a department-level web server at the University of Calgary (Department of Computer Science); a research group web server at University of Vigo (Signal

Processing Group); a campus-wide web server at the University of Saskatchewan; the EPA WWW server located at Research Triangle Park; the web server at NASA's Kennedy Space Center; the ClarkNet WWW server, an old commercial Internet provider in the Baltimore - Washington D.C. region; and the 1998 World Cup web site WWW server.

Log	Access Log Duration	Access Log Start Date	Access Log File (MB)	Total Requests	Requests per day
Calgary	1 year	Oct 24/94	49.8	726,739	2,059
Vigo	1 year	Jun 5/10	377	1,581,971	4,321
Saskatchewan	7 months	Jun 1/95	222	2,408,625	11,255
EPA	1 day	Aug 29/95	4.24	47,748	47,748
Nasa	2 months	Jul 1/95	355	3,461,612	56,748
Clarknet	2 weeks	Aug 28/95	327	3,328,587	237,756
World Cup	8 days	May 1/98	907	10,345,553	1,293,200

Table 3.3: Summary of Access Log Characteristics

In Table 3.3 we show a summary of the different servers' logs. We see that the number of requests per day varies by several orders of magnitude. As we see in Section 3.7 this has a great impact on the probability of false positives.

3.4.2 Results

According to [106], the inter-time between requests follows an exponential distribution. This implies that the requests follow a Poisson distribution with parameter λ that can be estimated using the Maximum Likelihood Estimator (MLE) [107]:

$$f_{Poisson(\lambda)}(k) = \frac{\lambda^k e^{-\lambda}}{k!}, \quad \text{and} \quad \hat{\lambda}_{MLE} = \frac{1}{N}\sum_{i=1}^{N} k_i, \tag{3.2}$$

where N is the number of considered samples.

Another distribution commonly used to model counting processes is the negative binomial distribution. We show the distributions in Table 3.4.

Distrib.	**pdf**
Poisson	$f(x\|\lambda) = \frac{\lambda^x e^{-\lambda}}{x!}$
Negat. Binomial	$f(x\|r,p) = \binom{x+r-1}{x}(1-p)^r p^x$
Binomial	$f(x\|n,p) = \binom{n}{x}p^x(1-p)^{n-x}$

Table 3.4: Pdf's of the candidate distributions for the number of requests.

To calculate the parameters we can use the MLE as follows [108]:

$$\hat{r}_{MLE} = \left\lfloor \frac{(\sum_{i=1}^{N} k_i)^2}{N\sum_{i=1}^{N} k_i^2 - (\sum_{i=1}^{N} k_i)^2 - N\sum_{i=1}^{N} k_i} + 0.5 \right\rfloor \tag{3.3}$$

$$\hat{p}_{MLE} = \frac{\sum_{i=1}^{N} k_i}{r \cdot N + \sum_{i=1}^{N} k_i}, \tag{3.4}$$

where N is the number of considered samples.

Log	Poisson MLE Param.	Neg. Bin. MLE Param.	Poisson K-L Div.	Neg. Bin. K-L Div.
Calgary	$\lambda = 0.024$	$p = 0.023$, $r = 1$	0.9104	0.0053
Vigo	$\lambda = 0.050$	$p = 0.048$, $r = 1$	0.8843	0.0350
Saskatchewan	$\lambda = 0.130$	$p = 0.115$, $r = 1$	0.6956	0.0046
EPA	$\lambda = 0.557$	$p = 0.358$, $r = 1$	0.3638	0.0018
Nasa	$\lambda = 0.684$	$p = 0.406$, $r = 1$	0.3018	0.0002
Clarknet	$\lambda = 2.753$	$p = 0.579$, $r = 2$	0.1147	0.0020
World Cup	$\lambda = 15.266$	$p = 0.836$, $r = 3$	1.2198	0.0051

Table 3.5: MLE Parameters and Goodness of Fit for modeling the logs

The estimated MLE parameters and the goodness of fit are shown in Table 3.5. The goodness of fit is measured using the KLD. From these results we can conclude that the negative binomial distribution is a better approximation than the Poisson distribution. This means that the data is overdispersed, i.e., the variance is larger than the mean.

Some daily and weekly trends in the logs can be observed in Figure 3.3. Our method ignores them and considers R as a stationary sequence. Also, it can be seen that the peak is only one-sample wide. Therefore, the requests can be considered uncorrelated.

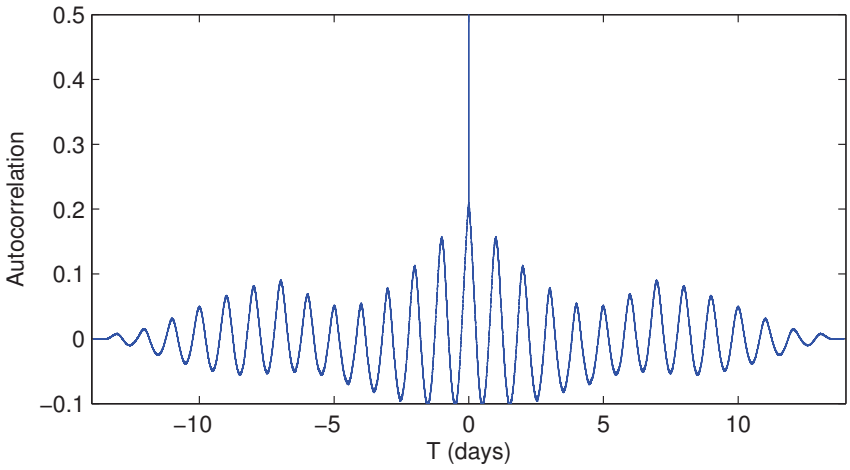


Figure 3.3: Autocorrelation of ClarkNet Log

We note that adding negative binomial distributed random variables with the same p results in another negative binomial distributed random variable whose parameter r is the sum of the respective r's [107]. In our problem we assume R to be a sequence of i.i.d. negative binomial random variables with parameters p_R and $w \cdot r_R$.

3.5 Fingerprinting Method

This section describes our algorithms to create and detect fingerprints.

3.5.1 Creating the Fingerprint

Ideally, we would like our fingerprint to be very difficult to detect by any other than its originator. Additionally, fingerprinting should be as fast as possible, in order to be able to read the fingerprint from a small fragment of the log file. We compute the rate of departures of our requests as $\lambda_{FP} = d \cdot \hat{\lambda}_S$, where d is the detectability factor, i.e., the increase in the number of requests the server will receive due to our

fingerprint (reasonable values are 0.01, 0.05, 0.1), and $\hat{\lambda}_S$ is a rough prediction of the rate of requests that the server receives.

When we want to fingerprint a server, we generate $n-1$ samples from an exponential distribution with rate λ_{FP}. These values are our IPDs, and the HTTP requests are sent to the server according to them. Therefore, the expected time to fingerprint the server is $(n-1)\lambda_{FP}$ seconds.

Note that we need to store the HTTP response "date fields" of the correct responses (status code 200), since this information will be later needed to recover the fingerprint. We denote the number of successful responses as n'.

3.5.2 Detecting the fingerprint

Detecting the fingerprint consists in deciding whether the log file contains the fingerprint. We propose two detectors: simple, that for each request decides if the fingerprint is present or not based on the number of requests inside the window, and the optimal likelihood-ratio test constrained to first-order statistics of the distributions.

3.6 Simple Detector

When it comes to deciding whether the ith request appears in the log or not, we look inside a window W_i containing $\hat{Y}_{x_i}$.

In Table 3.2 we observe that ε is always greater or equal to zero. We can conclude that W_i is reasonably given by $W_i = [\hat{Y}_{x_i}, \hat{Y}_{x_i}+w]$, for some integer $w \geq 0$. We define:

$$P_T(Y_{x_j}, W_i) \doteq Pr(Y_{x_j} \in W_i) = Pr(Y_{x_j} - \hat{Y}_{x_i} \leq w), \tag{3.5}$$

where $P_T(Y_{x_j}, W_i)$ can be interpreted as the probability that the log time of the jth request falls within the reference window of the ith request.

The detector considers the ith request from our fingerprint is present in the log when at least c_i entries in the log fall inside the window W_i. This can be mathematically expressed as $F_i = \sum_{j \in W_i}(E_j \geq c_i)$, where $c_i = \sum_{j=1}^{n'} \mathbb{1}_{W_i}(\hat{Y}_j)$. We denote the number of detected requests as $A = \sum_{m=1}^{n'} F_i$. We consider that a fingerprint is present in the log when we find at least Θ requests, i.e., $T \geq \Theta$

We note that this is not the optimal decoder, as the distribution of ε is not used. However, when ε is small, as it is the case considered in this problem (see Sect. 3.3), the proposed decoder is nearly optimal.

3.7 Analysis of the simple detector

In this section we calculate the theoretical probabilities of detection and false positives for the simple detector. Afterwards, we carry out some experiments to validate the results.

3.7.1 Probability of Detection

We recall that the number of requests per unit of time, S_j, is the sum of our (i.e., E_j) and other users' (i.e., R_j) requests. Furthermore, in Section 3.4 we showed that R^l can be modeled as an i.i.d. sequence distributed according to a negative binomial distribution with parameters p_R and $w \cdot r_R$.

Before delving into the analysis, we must model $E_{W_i} = \sum_{j=1}^{n'} \mathbb{1}_{W_i}(Y_{x_j})$, that represents the number of fingerprint entries on the log that appear inside W_i.

We know that the jth request has a probability of $P_T(Y_j, W_i)$ of appearing inside

W_i. Since E_{W_i} is the sum of Bernoulli random variables we can approximate it by a binomial distribution [109] with parameters:

$$n_k = \left\lfloor \frac{\left(\sum_{j=1}^{n'} P_T(Y_{x_j}, W_i)\right)^2}{\sum_{j=1}^{n'} P_T(Y_{x_j}, W_i)^2} + 0.5 \right\rfloor, \text{ and } p_k = \frac{\sum_{j=1}^{n'} P_T(Y_{x_j}, W_i)}{n_k}. \tag{3.6}$$

General Case

Here we study the probability of detection without making any assumption concerning λ_{FP}.

First, we study P_{D_i} that represents the probability that we correctly detect the ith request of our fingerprint in the log. Given that our detection algorithm is such that this event happens when we have at least c_i entries inside W_i, this probability is:

$$\begin{aligned}
P_{D_i} &= Pr(\sum_{j \in W_i} E_j + R_j \geq c_i) \\
&= \sum_{k=0}^{c_i-1} Pr(\sum_{j \in W_i} E_j \geq c_i - k) Pr(R_m = k) + Pr(\sum_{j \in W_i} R_j \geq c_i) \\
&\simeq \sum_{k=0}^{c_i-1} \left(1 - I_{1-p_k}(n_k - c_i + k + 1, c_i - k)\right) (1 - p_R)^{r_R(w+1)} \\
&\cdot \binom{l + r_R(w+1) - 1}{l} \cdot p_R^l + I_{p_R}(c_i, r_R(w+1)),
\end{aligned}$$

where $I_x(a, b)$ is the regularized incomplete beta function:

$$I_x(a, b) = \sum_{j=a}^{a+b-1} \binom{(a+b-1)}{j} x^j (1-x)^{a+b-1-j}.$$

Now we want to compute the probability that the number T of fingerprint entries found in the log is above the threshold Θ, which is when we declare the fingerprint

as detected. The random variable A is a sum of n' non-homogeneous dependent Bernoulli random variables, which we approximate by a binomial distribution [109] with parameters:

$$n_d = \left\lfloor \frac{(\sum_{i=1}^{n'} P_{D_i})^2}{\sum_{i=1}^{n'} P_{D_i}^2} + 1/2 \right\rfloor, \text{ and } p_d = \frac{\sum_{i=1}^{n'} P_{D_i}}{n_d}.$$

Now we can give an approximation of the probability of detection:

$$P_D = P(A \geq \Theta) \simeq 1 - I_{1-p_d}(n_d - \Theta + 1, \Theta).$$

Low λ_{FP} approximation

When the probability that two successive requests from the desired user fall in the same window is very small, i.e., $Pr(\hat{Y}_{i+1} - \hat{Y}_i \leq w) \simeq 0$, we can assume that $c_i \approx 1 \; \forall i$. This simplifies the resulting equations. This happens when λ_{FP} is several orders of magnitude smaller than $1/w$ requests per second. In this case P_{D_i} takes the value:

$$P_{D_i} = P_T(Y_{x_i}, W_i) + (1 - P_T(Y_{x_i}, W_i)) \cdot I_{p_R}(1, r_R(w+1)),$$

and A becomes a sum of n' homogeneous independent Bernoulli random variables. Therefore, A is binomially-distributed and the probability of detection becomes:

$$P_D = P(A \geq \Theta) = 1 - I_{(1-P_{D_i})}(n' - \Theta + 1, \Theta).$$

3.7.2 Probability of false positives

In this subsection, we calculate theoretically the probability of false positives, first without any assumption, and then assuming that the rate of requests is small.

General Case

Here we study the probability of false positives without making any assumption.

The probability that c_i entries in the log appear in an interval of size w when no fingerprint is actually present is

$$P_{F_i} = Pr(\sum_{j \in W_i} R_j \geq c_i) = I_{p_R}(c_i, r_R(w+1)).$$

Again, A is the sum of n' non-homogeneous dependent Bernoulli random variables, which can be approximated by a binomial distribution [109] with parameters

$$n_f = \left\lfloor \frac{(\sum_{i=1}^{n'} P_{F_i})^2}{\sum_{i=1}^{n'} P_{F_i}^2} + 1/2 \right\rfloor, \text{ and } p_f = \frac{\sum_{i=1}^{n'} P_{F_i}}{n_f}.$$

Now we can give an approximation to the false positive probability:

$$P_F = P(A \geq \Theta) \simeq 1 - I_{1-p_f}(n_f - \Theta + 1, \Theta).$$

Low λ_{FP} approximation

As before, when the rate of requests is small, we can approximate $c_i \approx 1$, $\forall i$. This implies that P_{F_i} takes the value:

$$P_{F_i} = I_{p_R}(1, r_R(w+1)).$$

Again A becomes the sum of n' homogeneous independent Bernoulli random variables, which means it is binomially-distributed and the false positive probability becomes:

$$P_F = P(A \geq \Theta) = 1 - I_{(1-P_{F_i})}(n' - \Theta + 1, \Theta).$$

3.7.3 Simple Detector Results

In order to validate our theoretical analysis we created a scenario where we can measure P_D and P_F. We implemented a simulator which gives us the probabilities

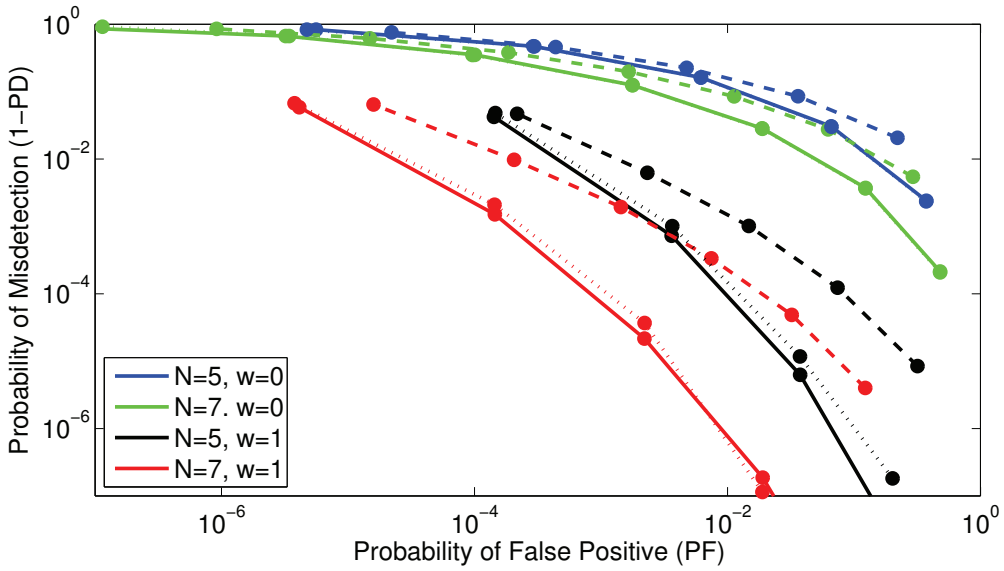


Figure 3.4: ROC of the simulations for Signal Processing in Communications research group's web server at the University of Vigo using the simple detector. Analytical (solid line), ideal conditions (dotted) and real conditions (dashed).

of detection and of false positives in two situations. The first is *ideal conditions* (i.e., R^n is a sequence of i.i.d. negative binomial random variables and ε^n is a sequence of i.i.d. random variables distributed according to Figure 3.2); the second is *real conditions*, where the log is taken from a web server and ε^n comes from the data of the last experiment in Section 3.3, preserving any existing autocorrelation.

Each experiment is carried out 10,000,000 times. We run this simulator for two different cases. The first one is the Signal Processing in Communications research group's web server at the University of Vigo where fingerprints are built by n=5 and n=7 entries. We consider two window sizes, w=0 and w=1 seconds. The second case is the web server at NASA's Kennedy Space Center. As it is a busier server, we make $n = 10$ to bring the false positive rates to acceptable levels, and we also use windows of size $w = 0$ and $w = 1$ seconds.

The results are shown in Figures 3.4 and 3.5, where we can see that the analytical

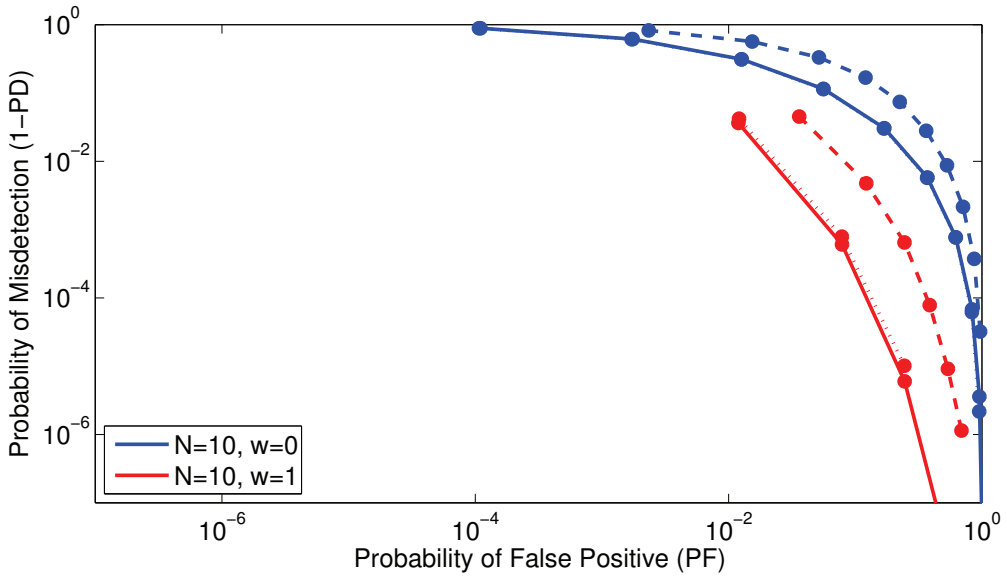


Figure 3.5: ROC of the simulations for the web server at NASA's Kennedy Space Center using the simple detector. Analytical (solid line), ideal conditions (dotted) and real conditions (dashed).

results closely match the simulated ideal conditions. This supports our theoretical analysis. We also see that only a discrete set of points (i.e., those marked with circles) are generated; this is due to the threshold Θ only taking integer values from 1 to n. Note that in some cases some points are missing, because it is not possible to measure those false positive probabilities (since we have run 10,000,000 experiments, probabilities under 10^{-7} could not be measured).

We can also see the performance loss due to non-ideal conditions. This is the gap between the dashed and the solid lines in Figures 3.4 and 3.5. The horizontal shift comes from assuming i.i.d. entries on the log and ignoring the trends, while the vertical shift comes from assuming independence on ε^n.

3.8 Optimal Detector

The detector previously proposed is suboptimal, as it uses neither the distribution of the error nor the distribution of other users' requests. We use the likelihood ratio test, that is proven to be optimal. However, for feasibility reasons, we have to restrict it to use only first order statistics.

This detector does not use a window, but the whole sequence S^l, as follows:

$$\Lambda(s^l, \hat{Y}_{x^n}) = \frac{Pr(S^l = s^l|H_1, \hat{Y}_{x^n})}{Pr(S^l = s^l|H_0)} = \frac{\sum_{e^l} Pr(E^l = e^l|\hat{Y}_{x^n}) Pr(R^l = s^l - e^l)}{Pr(R^l = s^l)} > \eta. \tag{3.7}$$

Assuming that R^l is an i.i.d. sequence the detector becomes:

$$\Lambda(s^l, \hat{Y}_{x^n}) = \frac{\sum_{e^l} Pr(E^l = e^l|\hat{Y}_{x^n}) \prod_{j=1}^{l} Pr(R_j = s_j - e_j)}{\prod_{j=1}^{l} Pr(R_j = s_j)} > \eta. \tag{3.8}$$

When we calculate $Pr(E^l = e^l|\hat{Y}_{x^n})$, for feasibility reasons we use first order statistics on ε^n, which is equivalent to assuming that ε^n is an i.i.d. sequence with pdf shown in Table 3.2.

3.8.1 Optimal Detector Results

We extend our simulator to use both detectors so we can compare their results. As the optimal detector is computationally more expensive, we run each experiment only 100,000 times. We run this simulator for two different cases previously presented: Scenario 1, the research group's web server at the University of Vigo with 5 requests, and Scenario 2, the web server at NASA's Kennedy Space Center with 10 requests. In both experiments, we use $\lambda_{FP} = 0.1$ requests per second.

The results are shown in Figures 3.6 and 3.7, where we can see that in the ideal case the optimal detector is expected to perform always better than the simple

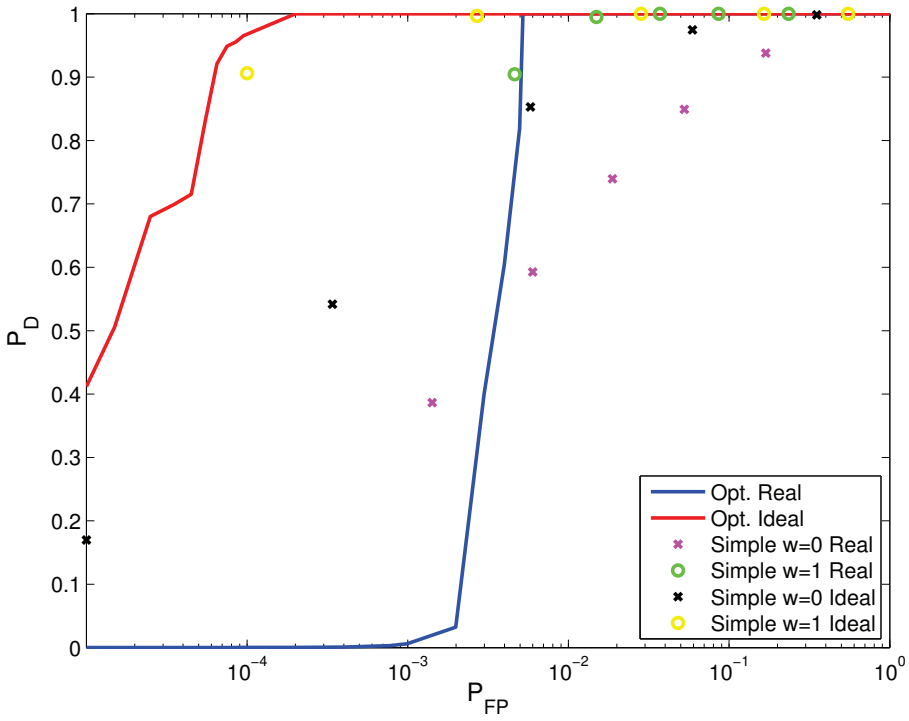


Figure 3.6: ROC comparison of both detectors for Signal Processing Group's web server at the University of Vigo $n = 5$.

detector, but in the real case, where some correlation exists, the optimal performs better in most of the situations, except at some points, where it is the opposite. This is due to the fact that the optimal detector considers only first order statistics in the prediction error as well as in other users' requests.

3.8.2 Computational Cost

Not only the performance is important for the detector but also the computational cost, to ensure that the algorithm is scalable and can be implemented in a real situation.

The complexity of the simple detector is linear, i.e. $O(n)$, but the optimal algorithm without any further assumption, has a complexity of $O(n^l)$ in the worst

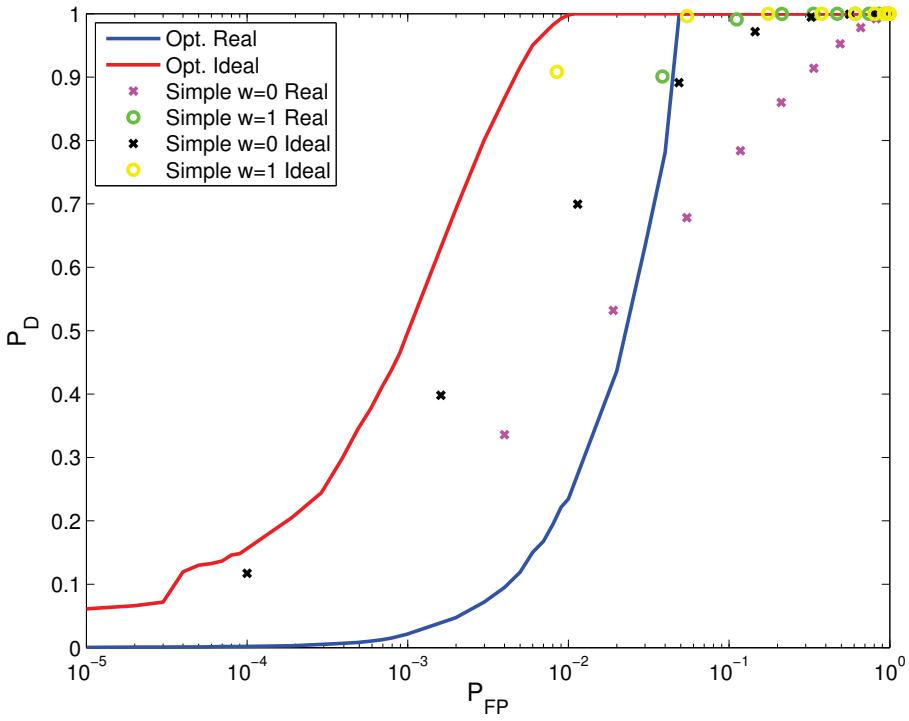


Figure 3.7: ROC comparison of both detectors for the web server at NASA's Kennedy Space Center $n = 10$.

case. This means that the optimal algorithm is not scalable at all, and it becomes infeasible for fingerprints with a length greater than 4. This algorithm can be simplified when the distribution of ε has a finite domain of width b. In this situation the optimal detector can be calculated in polynomial time of order b, i.e. $O(n^b)$. The results of the previous section where calculated truncating the distribution such that $Pr(\varepsilon > 3) = 0$, i.e., $b = 3$.

With the aim of comparing the computational cost of both detectors, we show in Table 3.6 their average running time to take a decision for the two scenarios of Section 3.8.1. The machine where this experiments were carried out is an 8-core Intel Core i7-3632QM CPU @ 2.20GHz with 8 GBytes of RAM.

Log	Simple Detector	Optimal Detector
University of Vigo	1653 μs	17654 μs
NASA's Kennedy Space Center	4725 μs	299437 μs

Table 3.6: Average Computational Time of simple detector and optimal detector

3.9 Conclusions

This chapter proposes a method to leave a timing fingerprint in the log of a hidden server. This fingerprint can be used as evidence that the server indeed hosted a particular content even after this content has been deleted. We note that, although our experiments have been carried out on a Tor hidden server, the underlying principles of the attack are valid for any web server.

Our approach is based on sending HTTP requests to the hidden server and storing the "date field" of the responses we get. We proposed two detectors: the first one checks whether there is a logged entry in a time window centered at the time that appears in the responses. Then, if the number of entries goes above a pre-defined threshold it decides that there is a fingerprint in the log. The second one is the optimal detector, which constructs a likelihood test based on first-order statistics.

We showed that the performance improvement is small comparing to the increase in the computational complexity, indicating that the theoretical optimal detector may not always be the best alternative, especially when the resources are limited or a scalable method is needed.

Chapter 4

Prediction-based Flow Correlation

4.1 Introduction

In this chapter we propose a first method to solve the problem set out in Section 2.3, i.e., deciding if an eavesdropped flow contains the flow of which we have the timing information. This first method assumes that an estimator of the arrival time of each packet or request is available.

In the proposed scenario, the traffic analyst (TA) tries to locate a hidden service from Tor network. We assume that the TA has access to the encrypted flow from the suspected machine to the Tor network. This adversary model has been shown to be realistic, as some organizations or governments have access to ISP (Internet Service Provider) data [110]. For that matter, the ISP itself or an AS (Autonomous System) on the path can become the TA.

The attack is done in the following way: the TA, that we call Alice, sends application requests to the server, that we call Bob, disguised as a common user's pattern. Alice has no problem to eavesdrop the flow as she is the source. Alice predicts the time that her packets will appear at Bob's side, but in order not to depend on Bob's

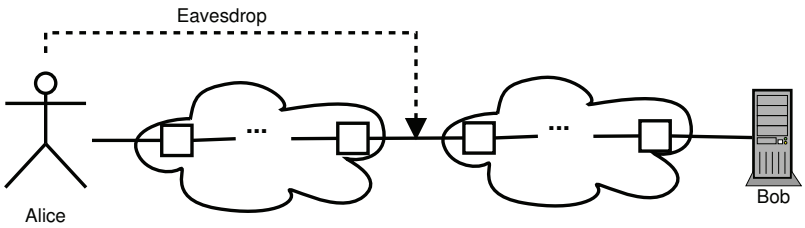


Figure 4.1: System Model

information, as in the previous chapter, she relies on the RTT (round-trip time). This prevents Bob from modifying this information. Finally, Alice makes a decision based on the likelihood-ratio test. This yields an excellent performance, for example Alice can identify a hidden service using the real traffic corresponding to the Signal Processing Group of the University of Vigo web server with just 8 requests achieving a detection probability 0.9 given a probability of false positive of 10^{-6}. Our flow-correlation attack can be considered as a passive analysis technique as we assume that Alice behaves exactly as a common user, thus making it impossible to detect using watermarking detection algorithms.

4.2 Description of the Problem

In this section, we formally describe the problem and recall the notation presented in Chapter 2.

4.2.1 Problem

Figure 4.1 shows the proposed application. Alice sends her traffic and has to decide whether the traffic she eavesdrops contains her flow or not. We also assume that the network can encrypt the traffic and repacketize the flow. This means that Alice cannot identify her packets by their size or contents.

Alice sends a fixed number of application messages following a pattern that could belong to a normal client, saving the time when she sends each request and the instant she receives the response. This is the interval considered for each request. Recall the definition of ith interval as the period of time from (2.7):

$$P_i = (b_i, e_i) = (b_i, b_i + T_i), \quad i = 0, \ldots, n-1, \tag{4.1}$$

in this application n is the number of messages that conform our flow. b_i is the instant that the ith request is sent and e_i is the instant that the response is received. To make intervals independent, Alice only sends a request when there are no other pending requests, i.e. (2.8) holds.

Alice predicts the value when she expects to see the message that carries her request at Bob's side. We denote as Y_{x_i} the time when the message x_i is received by Bob and $\hat{Y}_{x_i}$ the prediction of this time. The goal of Alice is to be able to correctly decide if the eavesdropped link contains the known flow based on the messages seen on this link.

We can define the following hypotheses:

H_0: The eavesdropped link does not carry Alice's flow.

H_1: The eavesdropped link carries Alice's flow.

In addition to obtaining a good performance, Alice wants to avoid that Bob or any other party different than Alice notices that she is trying to identify the flow, i.e., the sequence should have low detectability. As we have mentioned, our algorithm does not delay any packet and the sequence of requests follows a typical user pattern. Hence, the only possibility of being detected is due to an anomalous increment of traffic to Bob. We will measure the detectability with the KLD between the requests that the anonymous server receives with and without Alice's flow.

4.2.2 Proposed detector

The proposed detector is based on the likelihood-ratio test, $\Lambda(y^{n_2}, \hat{y}_{x^n})$, as follows:

$$\Lambda(y^{n_2}, \hat{y}_{x^n}) = \frac{f_{Y^{n_2}|\hat{Y}_{x^n},H_1}(t^{n_2}|\hat{y}_{x^n})}{f_{Y^{n_2}|H_0}(y^{n_2})}, \tag{4.2}$$

and Alice decides the eavesdropped link contains her flow if this ratio is larger than η, a threshold that we fix to achieve a certain probability of false positive.

We assume that messages coming from other users that fall inside an interval are uniformly distributed inside it (see [91]) and independent from Alice's message. In this case, the ratio can be simplified to

$$\Lambda(y^{n_2}, \hat{y}_{x^n}) = \prod_{i=1}^{n} \frac{f_\varepsilon(y_{x_i} - \hat{y}_{x_i})}{\frac{1}{T_i}} \tag{4.3}$$

where f_ε is the pdf of the prediction error. However, we find the problem that if Bob receives multiple messages in the interval we cannot identify a unique candidate to be the target message, that is, correctly measure the value of y_{x_i}.

We consider that the prior probability that any given message is the desired one to be equal for all messages. Hence, Alice decides that H_1 holds if

$$\Lambda(y^{n_2}, \hat{y}_{x^n}) = \prod_{i=1}^{n} \sum_{y_j \in P_i} \frac{1}{s_i} f_\varepsilon(y_j - \hat{y}_{x_i}) \cdot T_i > \eta. \tag{4.4}$$

where s_i is the number of messages that fall inside P_i. Note that since $\frac{s_i}{T_i}$ is the rate of requests, the larger this is, the less weight it is given in the decoding.

4.3 Application: Locating a Tor Hidden Service

In order to show the feasibility and usefulness of our algorithm we apply it to locate a hidden service. We assume a client, Alice, who sends a flow of messages to a hidden service, Bob, and eavesdrops the communication channel at Bob's side trying

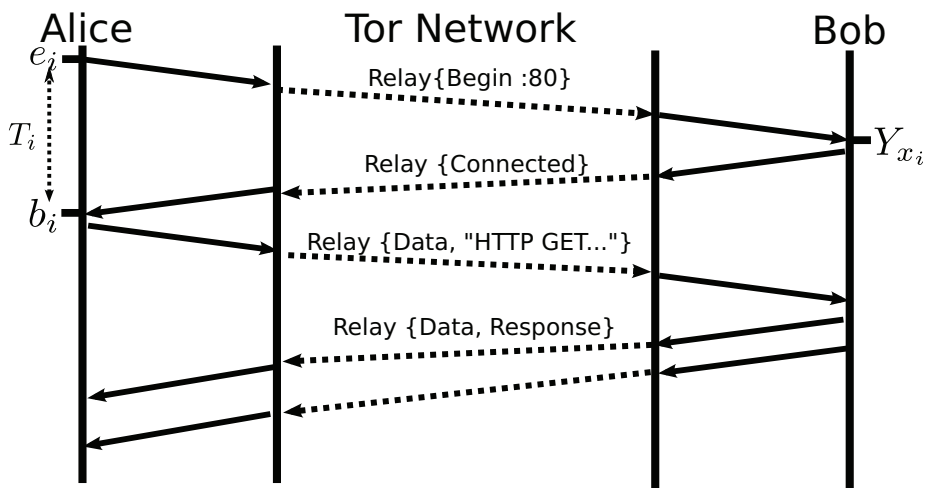


Figure 4.2: Cells in a Tor Request

to detect if it contains her flow. Figure 4.2 shows the packet exchange for just one request and the considered interval.

4.3.1 Predictor

Our algorithm benefits from a prediction of the time when a cell reaches the hidden service. The estimation is done based on the information available to Alice, that is, the RTT. Therefore, $\hat{Y}_{x_i} = f(T_i) + b_i$ where f is the prediction function.

Measured delays

In order to measure the delays, we created a toy client-server application to measure packet delays in Tor. Traffic is captured and matched to the packets shown in Figure 4.2. We used 50,000 samples that were taken every 10 seconds. As it is customary, we separate these data into two subsets: a training set that includes 80% of the samples, and the remaining 20% as test set.

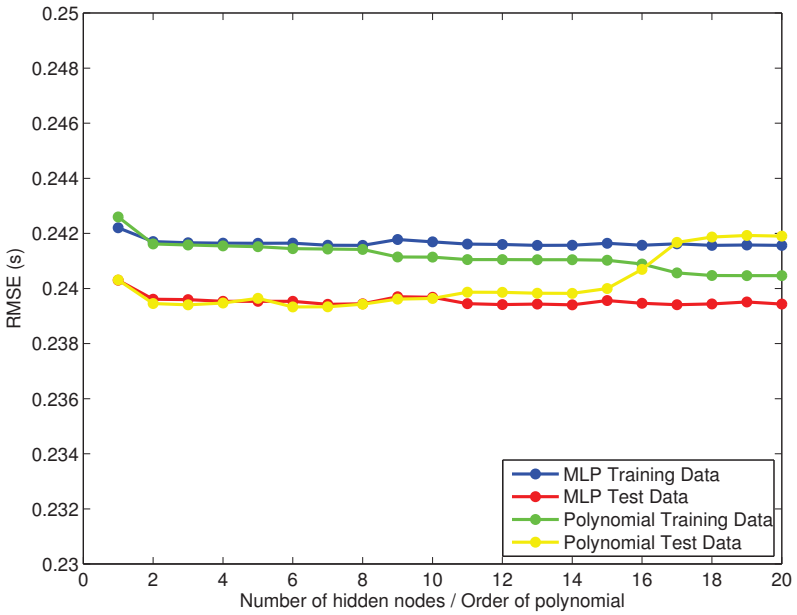


Figure 4.3: Performance of the MLP predictor and polynomial predictor

Predictors and performance

We propose two different predictors. The first one is based on polynomial regression and the second one a Multilayer Perceptron (MLP) [111]. The first model is simpler, but limits the range of available functions to predict, compared to the second one that can approximate any function [111].

We compare in Figure 4.3 the root mean square error (RMSE), as it is a measure of the accuracy of a predictor, for both cases. Results show that a MLP does not give any advantage and increasing the order of the polynomial produces negligible improvements. Hence our predictor is an affine function of T_i, that is, $\hat{Y}_{x_i} = 0.46 \cdot T_i + 0.02 + b_i$ with T_i measured in seconds. That is close enough to the expected $\hat{Y}_{x_{ii}} = 0.5 \cdot T_i + b_i$, as the paths of the request and response are essentially identical.

Distribution	**pdf**
Normal	$f(x\|_{\mu,\sigma}) = \frac{1}{\sigma\sqrt{2\pi}} \exp\left(-\frac{(x-\mu)^2}{2\sigma^2}\right)$
Cauchy	$f(x\|_{\mu,\sigma}) = \frac{1}{\pi\sigma\left(1+(\frac{x-\mu}{\sigma})^2\right)}$
Laplace	$f(x\|_{\mu,\sigma}) = \frac{1}{2\sigma} \exp\left(-\frac{\|x-\mu\|}{\sigma}\right)$
Logistic	$f(x\|_{\mu,\sigma}) = \frac{\exp\left(-\frac{x-\mu}{\sigma}\right)}{\sigma\left(1+\exp\left(-\frac{x-\mu}{\sigma}\right)\right)^2}$
Gumbel	$f(x\|_{\mu,\sigma}) = \frac{1}{\sigma} \exp\left(-\frac{x-\mu}{\sigma} - \exp\left(-\frac{x-\mu}{\sigma}\right)\right)$

Table 4.1: Distribution candidates for the prediction error

Characterizing the prediction error

We saw previously that the decoder needs to model the error function. To this end, we use the errors produced by the affine predictor given above. Errors are normalized dividing by their corresponding T_i.

We select some candidate distributions and estimate their parameters using MLE, as this method chooses the value of the parameters that produce a distribution that gives the observed data with the greatest probability. Afterwards, we evaluate the goodness-of-fit using different metrics: the KLD, the χ^2 , the Kolmogorov-Smirnov (KS) and the Anderson-Darling (AD). The smaller the value the better is the fit in all of them.

The candidate distributions were selected among the most common continuous distributions that have support at least on the range of our data, i.e $(-1, 1)$. The chosen distributions are Normal, Cauchy, Laplace, Logistic and Gumbel. Their pdfs can be seen in Table 4.1.

Table 4.2 shows the parameters obtained by MLE. Table 4.3 shows the goodness-of-fit test results, where it is shown that the Laplace distribution best models our data.

We also denormalize the intervals, hence we get the distribution of the error

Distribution	Parameters
Normal	$\mu = 0.0023$, $\sigma = 0.1474$
Cauchy	$\mu = 0.00048$, $\sigma = 0.0809$
Laplace	$\mu = 0.00014$, $\sigma = 0.1107$
Logistic	$\mu = 0.00013$, $\sigma = 0.0797$
Gumbel	$\mu = -0.0791$, $\sigma = 0.1983$

Table 4.2: MLE Parameters

	Test Filtered Data			
Distribution	**KLD**	χ^2	**KS**	**AD**
Normal	0.1608	801057	0.1005	1090
Cauchy	0.0846	586019	0.0763	197
Laplace	0.0315	592904	0.0310	176
Logistic	0.0888	600276	0.06545	514
Gumbel	0.3043	1540593	0.1561	17761

Table 4.3: Goodness of Fit

given the measured RTT, T_i, and we make the simplification $\mu \approx 0$ (cf. Table 4.2). Therefore, we can write

$$f_{\varepsilon_i|T_i}(\varepsilon_i|t_i) = \frac{1}{2\sigma t_i} \exp\left(-\frac{|\varepsilon_i|}{\sigma t_i}\right), \tag{4.5}$$

where the value of σ is shown in Table 4.2.

4.4 Analysis and Results

4.4.1 Mathematical analysis

In this section we want to calculate the threshold η for a given probability of false positive, and the probability of detection that is achieved with such threshold. First, we statistically model the number of cells per unit of time s_i/T_i and the round-trip time for a cell, T_i. Afterwards, we derive the expressions for η and P_D.

Distribution	**pdf**
Gen. Poisson	$f(x\|_{\lambda,\theta}) = \begin{cases} \frac{\lambda(\lambda+\theta x)^{x-1}\exp(-\lambda-\theta x)}{x!} & x = 0, 1, \dots \\ 0 & x > \lceil -\frac{\theta}{\lambda} \rceil \text{ if } \theta < 0 \end{cases}$
Gen. Neg. Bin.	$f(x\|_{p,\beta,r}) = \frac{r}{r+\beta x}\binom{r+\beta x}{x} p^x (1-p)^{r+\beta x - x}$

Table 4.4: Distribution candidates for the number of cells per unit of time

Modelling the number of cells per unit of time

We consider four possible models: the usual ones of chapter 3, Poisson and Negative Binomial, and their generalizations: Generalized Poisson and Generalized Negative Binomial. We show the formal pdfs in Tables 3.4 and 4.4 and we refer the reader to [112] for properties and parameter estimation.

To validate each model we use seven different World Wide Web server logs from the Internet Traffic Archive and UVigo. We estimate the parameters by MLE [112]. Results (cf. Table 4.5) show that both the Generalized Poisson and the Generalized Negative Binomial can model the number of requests. We choose to use the General Poisson Approximation as it has one degree less of freedom than the General Negative Binomial.

Log	Poisson K-L Div.	NB K-L Div.	Gen. Poisson K-L Div.	Gen. NB K-L Div.
Calgary	0.0121	0.0076	0.0001	0.0001
UVigo	0.0845	0.0504	0.0012	0.0012
Saskatchewan	0.0246	0.0066	0.0014	0.0017
EPA	0.0979	0.0025	0.0003	0.0001
Nasa	0.0882	0.0002	0.0001	0.00005
Clarknet	0.2534	0.0028	0.0009	0.0009
World Cup	1.9701	0.0074	0.0059	0.0056

Table 4.5: Goodness of Fit for the number of requests

Modelling a cell round trip delay

A characterization of the pdf of the round-trip time is needed. Our measurements confirm the result by Loesing et al. that the delays can be modeled as a Fréchet distribution [104].

Theoretical probabilities

Recalling from previous sections, we decide that the eavesdropped flow contains Alice's if the random variable W defined as

$$W = \prod_{i=1}^{n} \underbrace{\frac{1}{s_i} \sum_{Y_j \in P_i} \underbrace{\exp\left(-\frac{|Y_j - \hat{Y}_{x_i}|}{\sigma T_i}\right)}_{U_j}}_{V_i} \tag{4.6}$$

is larger than η. Also notice that in (4.6) we have defined the auxiliary random variables U_j and V_i.

Recall that Y_{x_i} represents the message coming from Alice's flow if it exists, while Y_j, $j \neq x_i$ corresponds to messages from any other source, then, for the latter kind of messages:

$$f_{U_j|T_i}(u_j|t_i) = \begin{cases} \frac{\sigma}{u_j} & u_j \in (a, b) \\ \frac{2\sigma}{u_j} & u_j \in (b, 1) \end{cases}, \; j \neq x_i \tag{4.7}$$

where

$$a = \min\left\{\exp\left(-\frac{0.46t_i + 0.02}{\sigma t_i}\right), \exp\left(-\frac{0.54t_i - 0.02}{\sigma t_i}\right)\right\}$$

$$b = \max\left\{\exp\left(-\frac{0.46t_i + 0.02}{\sigma t_i}\right), \exp\left(-\frac{0.54t_i - 0.02}{\sigma t_i}\right)\right\}.$$

For the case $j = x_i$, we have $f_{U_{x_i}|T_i}(u_{x_i}|t_i) \sim \text{Uniform}(0, 1)$.

We characterize V_i as

$$f_{V_i}(v_i) = \int_0^\infty \sum_{s_i=0}^{\infty} f_{V_i}(v_i|s_i,t_i) f_{S|T}(s_i|t_i) f_{T_i}(t_i) dt_i, \tag{4.8}$$

where $f(s_i|t_i)$ and $f_{T_i}(t_i)$ have been characterized in previous sections as Generalized Poisson and Fréchet distributions, respectively. The pdf $f_{V_i}(v_i|s_i,t_i)$ can be computed by convolving the distributions of $V_j|T_i$, for $j = 1,\ldots,n_i$. Formally,

$$f_V(v_i|s_i,t_i) = \begin{cases} \delta(v_i) & s_i = 0 \\ s_i \left(f_{U_0|T_i} * f_{V_1|T_i} * \cdots * f_{V_{n_i}|T_i} \right)(v_i s_i) & s_i > 0 \end{cases}. \tag{4.9}$$

Note that the case $s_i = 0$ is not possible when H_1 holds because the cell from Alice's flow must arrive inside the interval P_i. Also note that the convolution is evaluated at $v_i s_i$, this fact comes from (4.6) where the sum is multiplied by $1/s_i$.

Lastly, we characterize W, as $f(w) = f_{v_1 \cdot v_2 \cdots v_n}(w)$, where the density of the product of two independent random variables can be obtained as

$$f_{v_1 \cdot v_2}(v) = \int_v^1 \frac{1}{x} f_{v_1}(x) f_{v_2}(\frac{v}{x}) dx. \tag{4.10}$$

So we calculate the value of the threshold, η, as the $(1-P_F)$th quantile of $f_{w|H_0}$ and the probability of detection as $P_D = 1 - F_{w|H_1}(\eta)$, where F_w denotes the cdf of f_w. As an example, Figure 4.4 shows the theoretical P_D as a function of L for Uvigo traffic and $P_F = 10^{-6}$.

4.5 Results

We have created a simulator to show how close real results are from our predictions, and to compare with other existing approaches. Then we implement the proposed flow correlation scheme in the live Tor network against a hidden service.

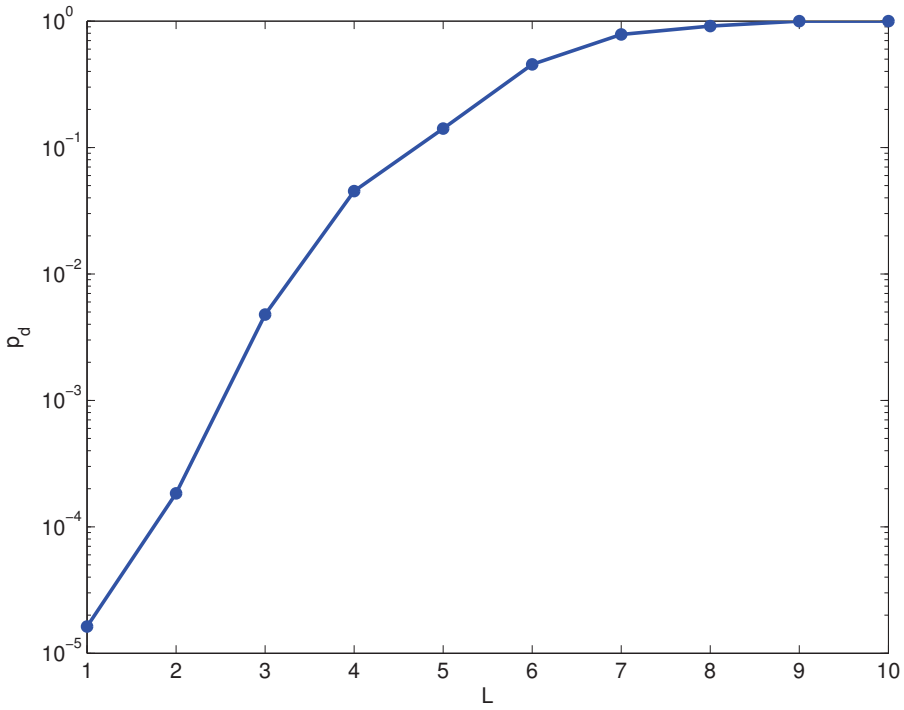


Figure 4.4: Theoretical P_D for UVigo log for $P_F = 10^{-6}$

4.5.1 Simulator

We created a simulator to validate our theoretical analysis and to compare with other existing approaches: RAINBOW [88], SWIRL [89] and Interval-based [90]. We simulate the following scenario. We send an HTTP request every 10 seconds, each request generates two cells (cf. Figure 4.2). Those flows are watermarked with each scheme. Each cell is delayed by an amount selected from the dataset so that the time correlation is preserved. To model the behavior of other users, we send HTTP requests mimicking the pattern that was measured in one of the logs.

Each experiment is simulated 10 million times. We run this simulator in two different scenarios. The first case assumes that the other users' requests are the same as for UVigo with 3 HTTP requests. The second scenario simulates the traffic of the web server at NASA's Kennedy Space Center. This second server is considerably

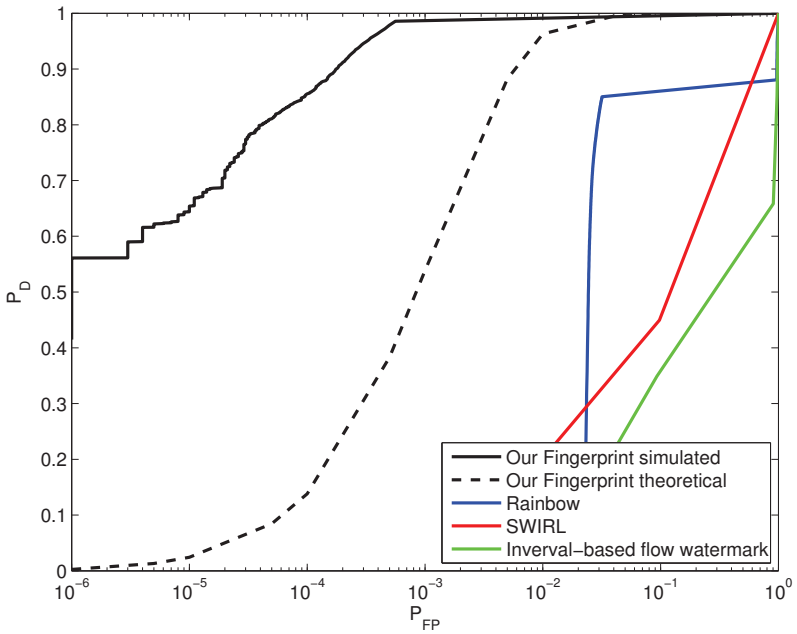


Figure 4.5: Simulator Results for UVigo web server with 3 requests

busier than the first one, so we increase the number of HTTP requests to 20.

Results are shown in figures 4.5 and 4.6. We see that this small number of requests is enough to achieve a very good performance. The performance of our algorithm, measured by the P_D for a given P_F is several orders of magnitude better than other watermarking schemes for the same number of messages. We also see that simulation results give a better performance than the analysis. This is due to the fact that in our analysis we assumed no autocorrelation in the log and in the delays, an assumption that does not hold in reality.

4.5.2 Real Implementation

Obviously, simulations are not fully realistic. For instance, our simulator assumes that no cells reach the hidden service other than the two generated by each request, which may lead to overestimating performance. This means that we can filter off

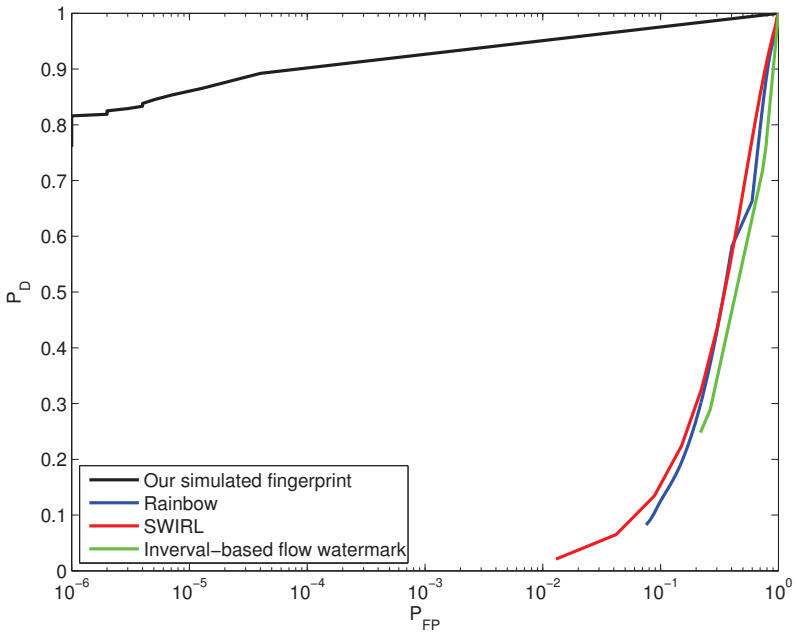


Figure 4.6: Simulator Results for NASA web server with 20 requests

all the control cells (i.e. padding, create and destroy [33]) and keep only those relay cells that carry actual information.

The real implementation was done using three computers connected to live Tor: a hidden service and two clients, one that tries to correlate the flow and another that sends requests according to the log of the simulated machine. We do not perform any kind of filtering, keeping all the control cells.

In our experiment, Alice sends one request 10 seconds after she receives the response. The gap between two different flow-correlation attacks is fixed to 30 seconds. We repeat 1,000 times the flow-correlation attack for each experiment. The experiment uses the UVigo log with $n = 1, 2$ requests. Results are shown in Figure 4.7. We see that the lack of filtering reduces the performance of the real implementation compared to the simulator and gives very similar results to the theoretical ones.

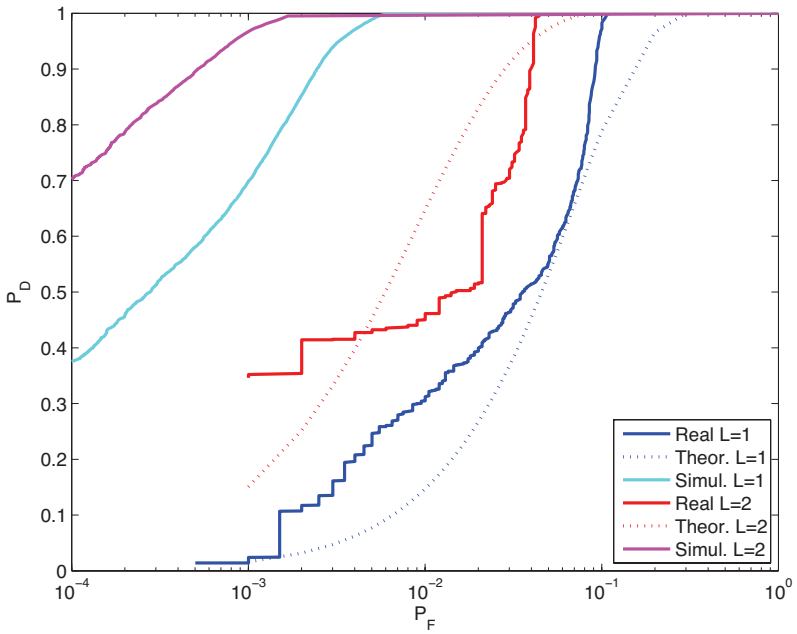


Figure 4.7: Real Implementation, Simulation and Theoretical Results for Uvigo web server

4.5.3 Detectability

We have mentioned that our scheme is completely undetectable through the TCP's intrinsic features as they are not modified, i.e. we do not add any extra delay to any message. Therefore, actual watermark detection algorithms from 2.4.3 cannot detect our flow-correlation attack as they rely on those characteristics. This also makes impossible that an intermediate node detects that Alice is trying to correlate the flows.

We also want to prevent that Bob can detect the flow-correlation attack, i.e. we want a low detectability. There are two ways Bob could detect our watermark: the first, due to the modification of the pdf of the number of received messages, and the second, due to the uncommonness of the messages pattern. We do not consider the second, as we have assumed that Alice's messages follow a normal user's pattern.

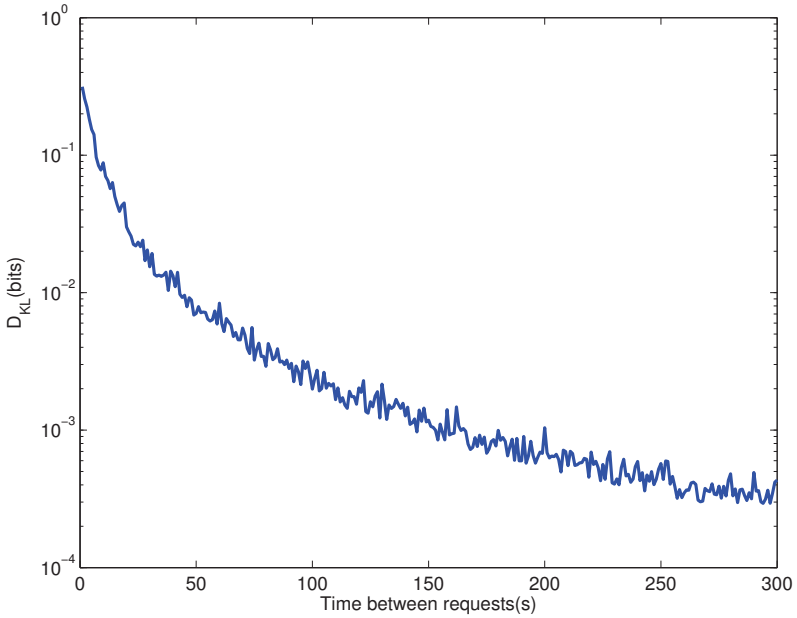


Figure 4.8: Detectability using the KLD

We measure the detectability using the KLD of the distributions of requests including Alice's flow and in its absence. Note that the detectability decreases with the time between requests, T_{req}, as seen in Figure 4.8, but we also want that the flow-correlation attack is done in a reasonable time.

Notice that the anonymous server may try to detect the watermark using higher-order statistics. Specifically, an increment of the autocorrelation periodically in $T_{req}+E[T_i]$ indicates the presence of this watermark as shown in Figure 4.9. This problem can be easily solved by making the time between requests pseudorandom as seen in Figure 4.10.

4.6 Conclusions

This chapter proposes a non-blind fingerprint for a flow created by the client. Our proposed scheme outperforms existing methods. This is due to two reasons: first,

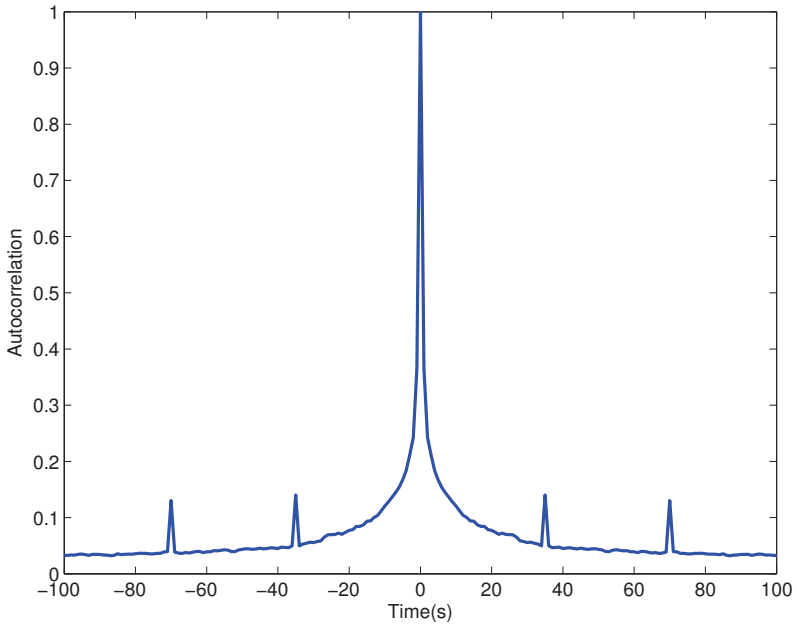


Figure 4.9: Autocorrelation of the number of requests per second with time between requests fixed to 30

information generated during the creation of the fingerprint is used in the detection, and second, the use of an optimal decoder.

The fingerprint is constructed by sending requests, each request determines one interval. A prediction of the time of arrival is done for each request. The performance is studied theoretically and empirically, through both a simulator and a real implementation of the algorithm. Results show that we can create a fingerprint with very few requests: less than 10 for a server with little traffic and of a few tens for a busier web server.

An implementation of the attack against a real hidden server connected to the Tor network has been carried out, showing a performance greater than the theoretical results.

We also study the detectability of the algorithm, and see that the larger the average time between requests, the less detectable our algorithm is, and that we

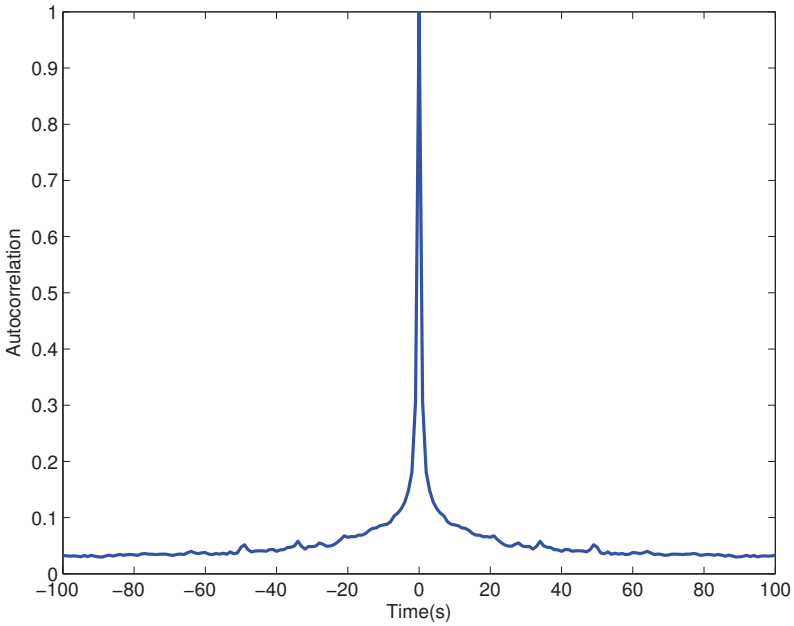


Figure 4.10: Autocorrelation of the number of requests per second distributed uniformly between 0 and 60

need the time between intervals to be pseudorandom to avoid detection through higher order statistics.

Chapter 5

Interval-count-based Flow Correlation

5.1 Introduction

In this chapter we deal with the same problem as in Chapter 4, i.e., deciding if an eavesdropped flow is carrying a flow of interest. This second method is based on detecting an increment of the number of packets that fall inside certain intervals with respect to the expected when the flow of interest is not present.

We use the same scenario as in Chapter 4, locating a hidden service from Tor network with the attacker having access to the encrypted flow from the suspected machine to the Tor network.

The attacker sends requests to the hidden service and the decision is made by detecting the increment of traffic in the eavesdropped link. Time is divided in intervals and the test considers those in which the cells of the request are predicted to reach the hidden service. Results show that we can identify a hidden service in a reliable way with just a couple of requests.

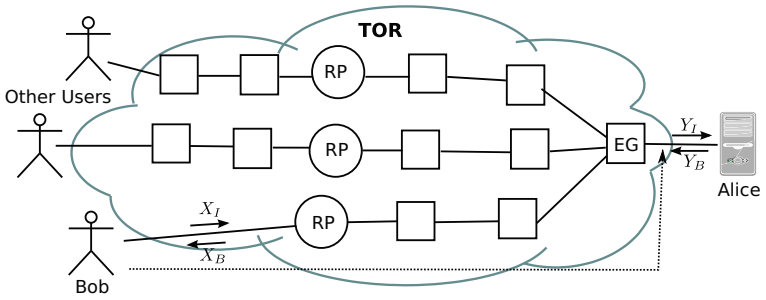


Figure 5.1: System Model

5.2 Model Problem

In this section, we formally describe the problem and recall the notation presented in Chapter 2.

The system model is shown in Figure 5.1, where Alice connects to Bob through a one-hop Tor circuit to the rendezvous point, to reduce the latency variability. Alice is also able to obtain the time and size of the packets from the flow between the suspected Bob and his entry guard. Alice wants to correctly decide whether that flow belongs to Bob or not.

Our application deals with two bidirectional cell flows: the flow that Alice sends to Bob through its rendezvous point with timing information X^n, and the eavesdropped flow (suspected to belong to Bob) with timing information Y^{n_2}. As the flows are bidirectional, we add a subscript I or B to differentiate the direction: I for the flow that goes from Alice to Bob and B for the opposite direction. In the case that the expression is valid for both directions, we drop this subscript to keep the notation simpler.

We consider the following intervals at Bob's side:

$$P_i = (b_i, e_i) = (b_0 + i \cdot T, b_0 + (i+1) \cdot T), \quad i = 0, \ldots, L-1, \tag{5.1}$$

where b_i is the beginning of each interval, e_i is the end, T is the interval length and L is the number of considered intervals. Alice chooses $b_0 = X_1$, i.e., the first interval starts at the moment Alice sends her first cell to Bob, and intervals are contiguous.

Alice makes her decision based on the number of cells that fall inside P_i, denoting this random variable by S_i, hence:

$$S_i = \sum_{j=1}^{n_2} \mathbb{1}_{P_i}(Y_j). \tag{5.2}$$

where $\mathbb{1}_A(x)$ is the indicator function of $x \in A$.

The flow at the alleged Bob can have either one of two sources: the cells from Alice's flow or a different source, i.e., other users' flows. We represent this as $S^L = E^L + R^L$, where E^L is the number of cells that come from Alice and R^L denotes the remaining cells. Formally,

$$E_i = \sum_{j=1}^{n} \mathbb{1}_{P_i}(Y_{x_j}). \tag{5.3}$$

where Y_{x_j} indicates the value of Y that corresponds to the same cell as X_j, if it exists.

Formally, we can express this problem via classical hypothesis testing with the following hypotheses:

H_0: The suspected flow does not contain Alice's flow.

H_1: The suspected flow contains Alice's flow.

5.3 Basic Detector

In this section we derive our detector and model the distributions of the number of packets from other sources and the network delay.

5.3.1 Detector construction

As in previous chapters, we use the likelihood ratio test for constructing our detector in order to get the maximum probability of detection for a given probability of false positive. Hence, our detector chooses H_1 when

$$\begin{aligned}\Lambda(s^L|x^n) = \frac{\mathcal{L}(H_1|S^L, X^n)}{\mathcal{L}(H_0|S^L)} &= \frac{f(s_B^L|x_B^n, H_1)}{f(s_B^L|H_0)} \frac{f(s_I^L|x_I^n, H_1)}{f(s_I^L|H_0)} \\ &= \frac{\sum_{e^L} f_{E_B^L}(e^L|x_B^n) f_{R_B^L}(s_B^L - e^L)}{f_{R_B^L}(s_B^L)} \\ &\cdot \frac{\sum_{e^L} f_{E_I^L}(e^L|x_I^n) f_{R_I^L}(s_I^L - e^L)}{f_{R_I^L}(s_I^L)} > \eta \end{aligned} \tag{5.4}$$

and H_0 in the opposite case. Recall that $\mathcal{L}$ represents the likelihood function and η is a threshold that we fix to achieve a certain probability of false positive. Note that R has to be greater or equal to 0, hence e_i can never be larger than s_i.

For feasibility reasons, we constraint the detector to use first-order statistics, discarding the information carried by higher-order statistics. We also assume that the amount of traffic from other sources, R^L, is independent of Bob's flow.

Hence the likelihood ratio becomes

$$\begin{aligned}\Lambda(s^L|x^n) = \prod_{i=1}^{L} &\frac{\sum_{e_i=0}^{s_{B,i}} f_{E_{B,i}|X_B^n}(e_i|x_B^n) f_{R_B}(s_{B,i} - e_i)}{f_{R_B}(s_{B,i})} \\ &\cdot \frac{\sum_{e_i=0}^{s_{I,i}} f_{E_{I,i}|X_I^n}(e_i|x_I^n) f_{R_I}(s_{I,i} - e_i)}{f_{R_I}(s_{I,i})} > \eta. \end{aligned} \tag{5.5}$$

From (5.5), we notice that we need to model the number of cells a hidden service receives from other sources (i.e., other clients and control cells) and the number of cells we will receive in each interval from Alice's flow, i.e. determine $f_R(r)$ and $f_{E_i|X^n}(e_i|x^n)$ for both directions I and B.

5.3.2 Modelling the number of cells per unit of time

To model the distribution of the number of cells a hidden service receives per unit of time, we first measure them in several real connections, then fit these data to some candidate distributions and select the distribution that matches best.

Alice does not know the distribution of R so she uses the flow prior to sending her request as training set. We denote by T_{tr} the length of this training set. Since Alice uses the traffic just before sending her request, she minimizes the effects of the non-stationary traffic to Bob.

We used two hidden web servers that replicate two real ones: the old hidden service Silk Road (SR), and the search engine DuckDuckGo (DDG). We request each web page according to a Poisson model with rates 58.75 requests per hour for SR and 2.75 requests per hour for DDG. These vaues are chosen so the total number of requests is the measured in the original hidden service according to [113]. The data were captured during 24 hours.

To construct the model, we assume an i.i.d. (independent and identically distributed) sequence. We use the following models: the empirical distribution; the kernel density estimation (KDE), a non-parametric probability density function (pdf) estimator also called Parzen-Rosenblatt window; Poisson; Negative Binomial, and Generalized Poisson [112]. The pmfs of the parametric models, i.e., those that have a closed-form pmf, are shown in previous chapters in Tables 3.4 and 4.4.

To calculate the KDE we use a Gaussian kernel as shown in [114]. For the parametric distributions, i.e. Poisson, Negative Binomial and Generalized Poisson, we estimate the parameters through MLE.

We measure the goodness of fit between the test sequence and the model using the square root of the Jensen-Shannon divergence (JSD).

We depict the results in Figures 5.2, 5.3 for the SR hidden service and in Figures

5.4 and 5.5 for the DDG hidden service. First, we see that the Poisson model is not adequate for cells even when the users requests follow a Poisson distribution, which is consistent with the results of [115]: users' requests can be well-modelled as a Poisson process but packets cannot due to their burstiness. Second, it is clear that T_{tr}=5 minutes is enough to model R properly.

Although the empirical distribution gives the best fit, we decide not to use it because of its lack of robustness, for instance, if a certain value, r, does not appear in Alice's training set, then the empirical model would imply $f_R(r) = 0$. Among the rest we decide to use the Negative Binomial model as its performance is better than the other parametric models for both directions, I and B, and the KDE performs well in some situations but not always. Hence, we consider

$$f_R(r|t,q) = \binom{t+r-1}{r} q^t (1-q)^r \tag{5.6}$$

where t and q are the parameters that Alice obtains through MLE using the previous 300 seconds before sending her request, as previously indicated.

5.3.3 Modelling the number of cells from Alice's flow in each interval

In order to characterize the number of cells from Alice that appear in P_i, i.e., $f_{E_i|X^n}$, we fist predict the moment when a cell appears at Bob's side ($\hat{Y}_x$), then we characterize the prediction error with the aim of obtaining the probability of a concrete cell appearing in each interval, and finally we obtain a model for $f_{E_i|X^n}$.

Predicting cell time at the hidden service

Traffic from hidden services is bursty in most of the cases, such as the response of HTTP. This implies that the cell delay depends on the position inside the burst,

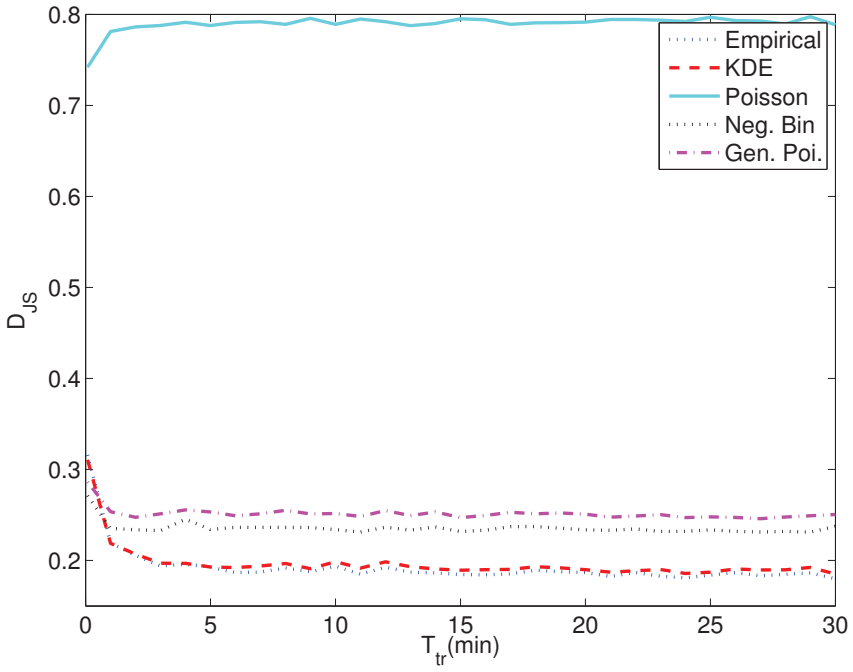


Figure 5.2: Goodness of fit for the different models for flow B to SR Hidden Service (T_B=100 ms).

thus we cannot consider cell delays to be identically distributed. To overcome this problem, we predict the cell delay, $\hat{Y}_{x_j}$, with information from Alice's flow including the position inside its burst.

Figure 5.6 shows the sequence of cells sent after the rendezvous connects the circuits from Alice and Bob, where besides the `RELAY_DATA` cells (shown in black) we plot the `RELAY_SENDME` (shown in red), that are used for stream-level flow control (Onion Proxy to Onion Proxy) and circuit-level flow control (Onion Router to Onion Proxy). These are sent in a deterministic way (in general, every 50 cells for stream-level flow control and every 100 cells for circuit-level flow control). Note that the cells in the B direction tend to scatter, in the sense that the Tor delay tends to increase.

We predict the values of $\hat{Y}_{X_{B,j}}$ as a function of RTT_0 (Round Trip Time with a burst of only 1 cell), RTT_n (Round Trip Time corresponding to the cell to predict)

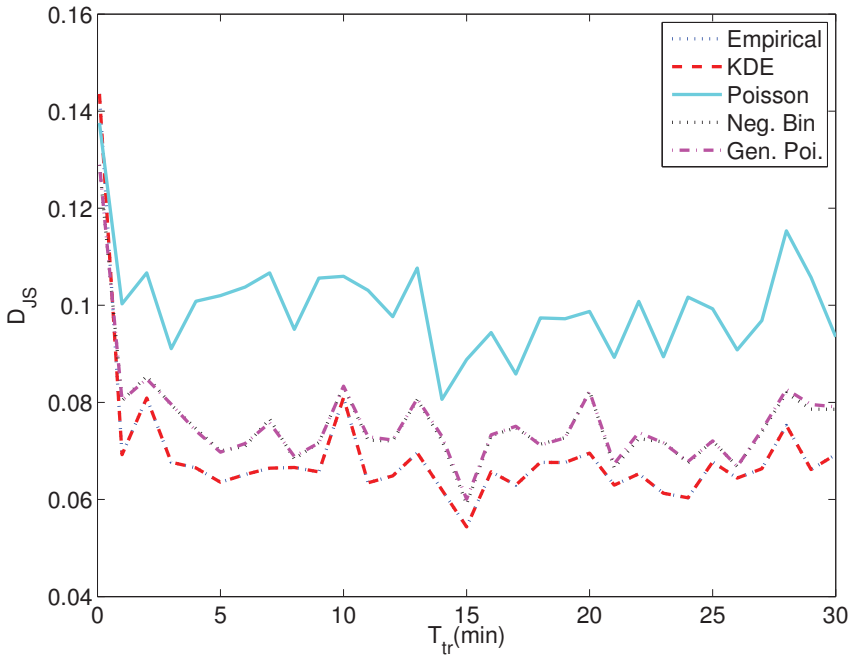


Figure 5.3: Goodness of fit for the different models for flow I to SR Hidden Service (T_I=100 ms).

and n (number of cells inside the burst). We show in Figure 5.6 how Alice measures these values. The values of $\hat{Y}_{X_{I,j}}$ are obtained as a function of only the RTT_0 as they do not have a bursty nature.

First, we deploy a hidden service that returns an object of the requested size and capture the traffic at both ends, i.e. Alice and Bob. We request objects of 1 KB, 10 KB, 100 KB and 500KB for a total of 500 of each kind using different circuits for each request. As it is customary, we separate these data into three subsets: a training set, an evaluation set and a test set, using 350 requests for the training set and 75 requests of each size for both the evaluation and test sets. We remove the circuit flow-control cells as they do not reach the other end.

For the flow I we use the predictor from Chapter 4, that is, $\hat{Y}_{X_{I,j}} = 0.46 \cdot \mathrm{RTT}_0 + 0.02 + X_{I,j}$.

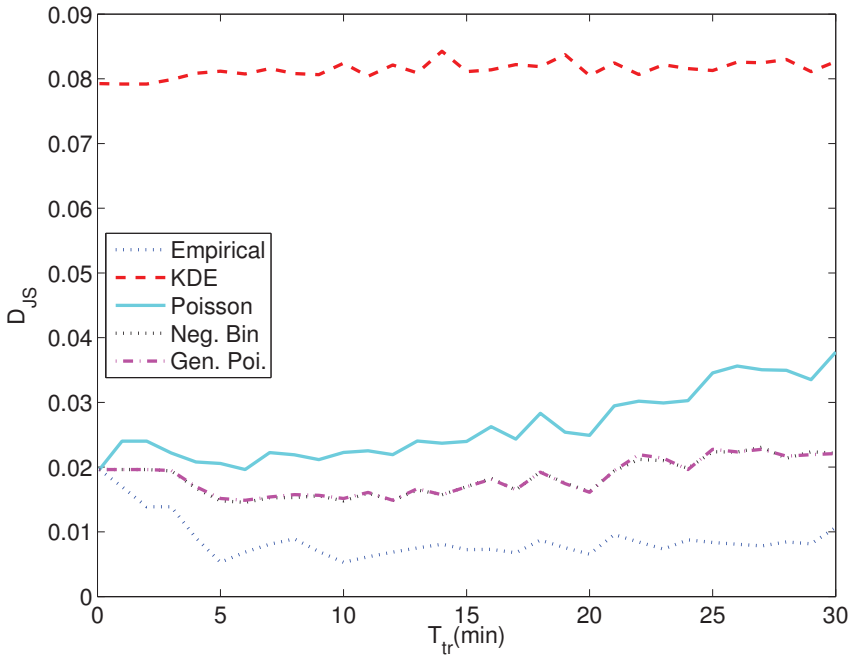


Figure 5.4: Goodness of fit for the different models for flow B to DDG hidden service (T_B=250 ms).

We compare in Figure 5.7 the accuracy of the predictor for flow B $\hat{Y}_{X_{B,j}}$ measured by the mean-square error (MSE). We decide to use a MLP with 5 neurons in the hidden layer as increasing the number of nodes above this value produces negligible improvements. Polynomial predictors of 6 and higher order give a larger MSE due to precision errors.

Characterizing the prediction error

We want to obtain the probability that a cell appears at Bob's side within a particular interval. The first step is characterizing the prediction error. This is done by fitting the errors made by the predictor of the previous section to some candidate distributions and selecting the distribution that matches best.

The candidate distributions are Cauchy, Gumbel, Laplace, Logistic and Nor-

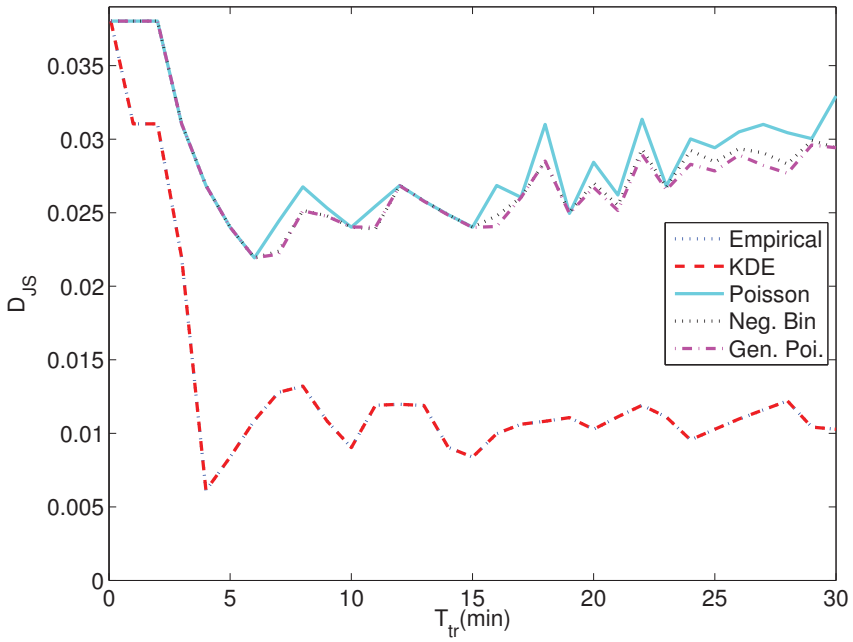


Figure 5.5: Goodness of fit for the different models for flow I to DDG hidden service (T_I=250 ms).

mal.Their pdfs are shown in Table 4.1 . We estimate the respective parameters using robust statistics to prevent that outliers coming from wrong cell identification affect the measurements, as explained in [116, Chapter 3].

Results from Table 5.1 show the estimated parameters and the goodness of fit using the JSD, being the Cauchy distribution the best fit.

Table 5.1: Goodness of fit of the candidate distributions for Prediction Error and its parameters

		Cauchy	Gumbel	Laplace	Logistic	Normal
Flow I	D_{JS}	0.1117	0.1784	0.1475	0.1793	0.1999
	μ_I	−0.0100	−0.0227	−0.0100	−0.0100	−0.0100
	σ_I	0.0266	0.0347	0.0384	0.0243	0.0395
Flow B	D_{JS}	0.1224	0.1937	0.1238	0.1568	0.1910
	μ_O	0.0135	−0.0164	0.0135	0.0135	0.0135
	σ_O	0.0627	0.0817	0.0904	0.0570	0.0929

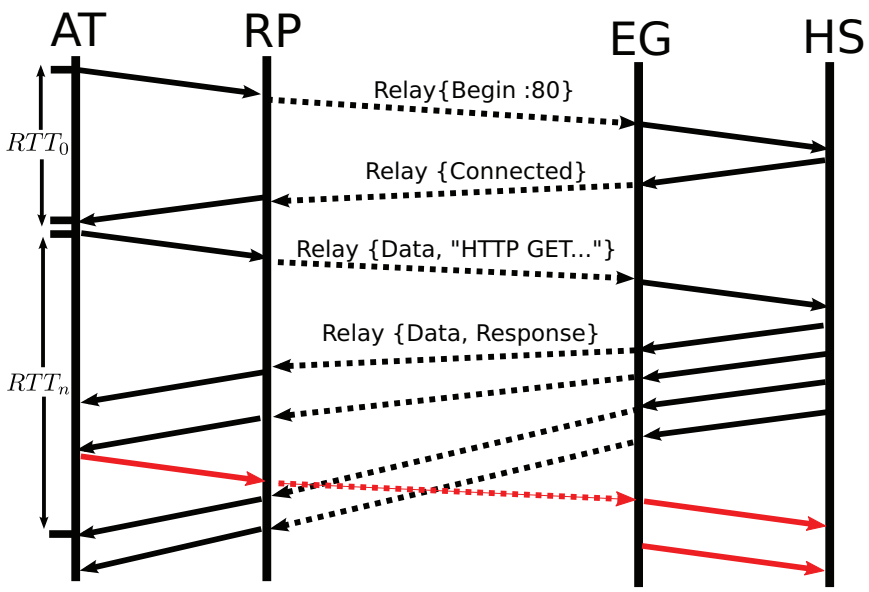


Figure 5.6: Cell Sequence of an HTTP request-response to a HS

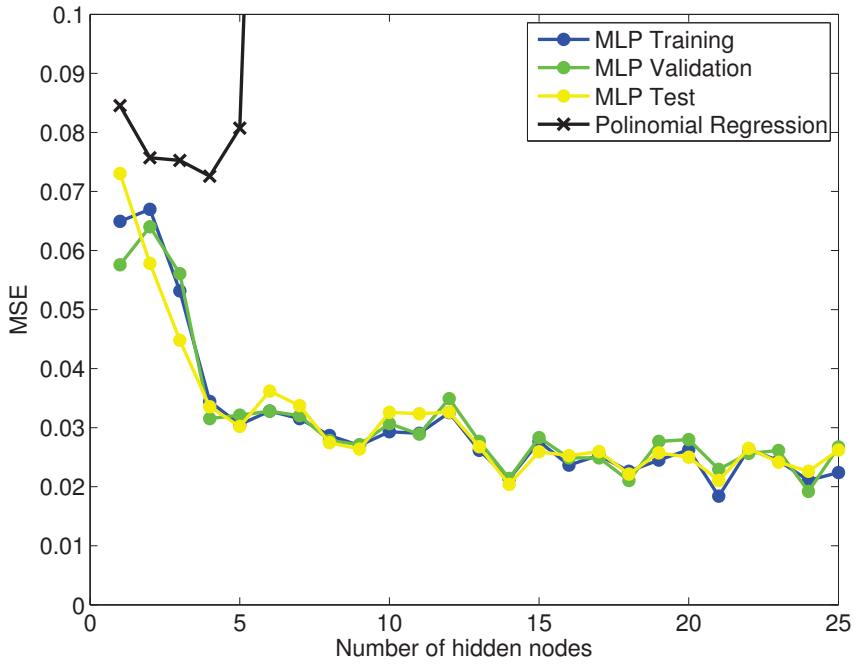


Figure 5.7: Performance of the MLP predictor and polynomial regression

Therefore, we can calculate the probability that the cell X_j leaves (B) or reaches (I) Bob's interval P_i as follows:

$$Pr(Y_{x_j} \in P_i) = \frac{1}{\pi}\left(\arctan\left(\frac{b_i + T - \hat{Y}_{x_j} - \mu}{\sigma}\right) - \arctan\left(\frac{b_i - \hat{Y}_{x_j} - \mu}{\sigma}\right)\right). \tag{5.7}$$

Model for the number of cells Bob receives from Alice in each interval

After obtaining the probability that the cell X_j appears at Bob's link within the interval P_i, we can determine $f_{E_i|X^n}(e_i|x^n)$ as a sum of n non-homogeneous dependent Bernoulli random variables. Formally, $E_i = \sum_{i=1}^{n} \text{Bernoulli}(Pr(Y_{x_j} \in P_i))$, which we can approximate by a binomial distribution [109] with parameters:

$$m_i = \left\lfloor \frac{\left(\sum_{j=1}^{n} Pr(Y_{x_j} \in P_i)\right)^2}{\sum_{i=1}^{n} Pr(Y_{x_j} \in P_i)^2} + 1/2 \right\rfloor \text{ and}$$
$$o_i = \frac{\sum_{i=1}^{n} Pr(Y_{x_j} \in P_i)}{m_i}. \tag{5.8}$$

Therefore,

$$f_{E_i|X^n}(e_i|x^n) = \binom{m_i}{e_i} {o_i}^{e_i}(1-o_i)^{m_i - e_i} \tag{5.9}$$

5.3.4 Detector

Hence, from (5.5) we obtain the likelihood-ratio test as:

$$\Lambda(s^L|x^n) = \prod_{k=1}^{L} \frac{f_{\left(\text{NB}(t_I,q_I)*\text{Bin}(m_{(k,I)},o_{(k,I)})\right)}(s_{(k,I)})}{f_{(\text{NB}(t_I,q_I))}(s_{(k,I)})} \cdot \frac{f_{\left(\text{NB}(t_B,q_B)*\text{Bin}(m_{(k,B)},o_{(k,B)})\right)}(s_{(k,B)})}{f_{(\text{NB}(t_B,q_B))}(s_{(k,B)})} > \eta \tag{5.10}$$

where $*$ denotes the convolution operation that can be expressed as:

$$f_{(\mathrm{NB}(t,q)*\mathrm{Bin}(m,o))}(x) = \sum_{j=0}^{\min(m,x)} f_{\mathrm{NB}(t,q)}(x-j) \cdot f_{\mathrm{Bin}(m,o)}(j) \tag{5.11}$$

where $f_{\mathrm{NB}(t,q)}$ and $f_{\mathrm{Bin}(m,o)}$ denote the pdfs of a negative binomial and binomial distribution, respectively. These distributions are shown in Table 3.4.

5.3.5 Calculating the threshold to achieve a certain probability of false positive

We first show how the threshold can be obtained theoretically, but in a real implementation we suggest to use a Monte Carlo simulation due to the fact that this theoretical calculation is not practical when L grows.

From the previous section, the Test (5.10) can be expressed as

$$\Lambda(s^L|x^n) = \overbrace{\prod_{k=1}^{L} \underbrace{g_I(s_{k,I})}_{U_I} \underbrace{\cdot g_B(s_{k,B})}_{U_B}}^{W} > \eta \tag{5.12}$$

where the function g is the ratio of the two likelihood functions in (5.10), i.e., f_{NB*B} and f_{NB}, but once the parameters m, o, t and q are fixed, it becomes deterministic. Hence, the distribution of $U|H_0$ can be obtained as $f_{U|H_0}(u) = \sum_{s\in g^{-1}(u)} f_{(\mathrm{NB}(t,q))}(s)$.

Finally, we characterize W as $f(w) = f_{U_{I,1}\cdot U_{O,1}\cdots U_{I,L}\cdot U_{O,L}}(w)$, where the density of the product of two independent random variables can be obtained as $f_{U_1\cdot U_2}(w) = \sum_{u_1,u_2:u_1u_2=w} f_{U_1}(u_1)f_{U_2}(u_2)$. So we can calculate the value of the threshold, η, as the $(1-P_F)$th quantile of $f_{w|H_0}$.

This method has the drawback that its complexity grows exponentially with L, so in a practical scenario we use a Monte Carlo method, assuming the model for R is accurate.

5.4 Results

In order to validate our proposal, we carried out an experiment on the live Tor network. We used the two hidden web services discussed in Section 5.3.2, i.e. DDG and SR. Besides Alice, 10 different machines are requesting the web page as explained in Section 5.3.2 so the total number of requests from those clients is the same as in the original hidden service measured by [113], that is, 27.5 and 587.5 requests per hour for DDG and SR, respectively. We captured the traffic on both ends with `tcpdump` and we repeated the experiment 1000 times.

5.4.1 Interval size

We first analyze the influence of T, the interval size, on the performance, using thresholds calculated with Monte Carlo methods as mentioned in the previous section.

In the case that our data would perfectly fit the model, the smaller the interval the best results we would expect to obtain. In practice, this may not be true; for instance the prediction error is likely to be correlated for two contiguous cells. This information cannot be inferred from first order statistics.

In order to measure the performance we use different values of T. We depict the result for the DDG hidden service in Figure 5.8 and in Figure 5.9 for the SR hidden service. For the DDG hidden service the detector performs better in a neighborhood of $T = 0.5$ s, but for SR it seems that the smaller the interval, the better. Recall that the the threshold is calculated assuming that R comes from an i.i.d negative binomial whose parameters are obtained through MLE using the previous 5 minutes of the suspected flow, as explained above.

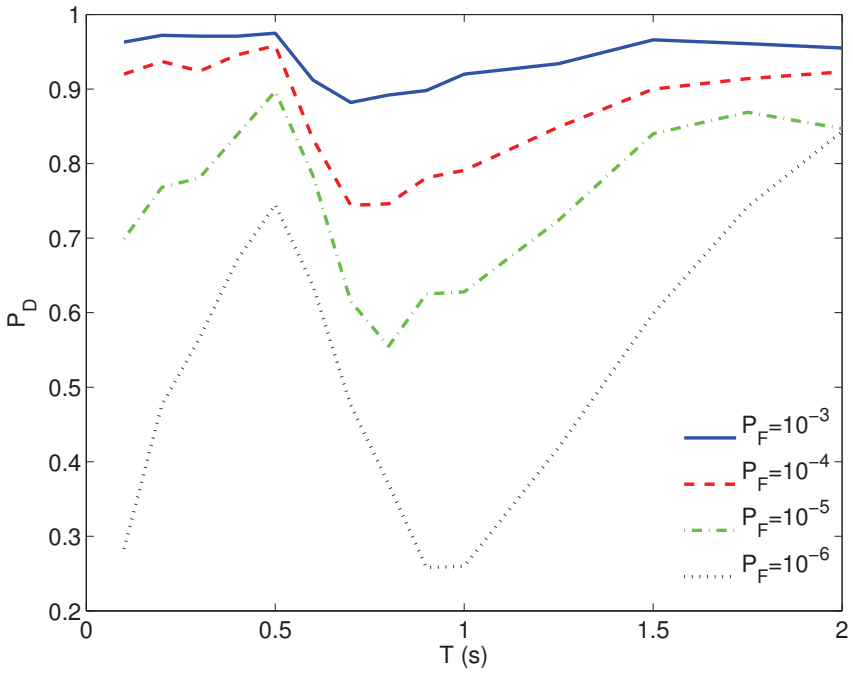


Figure 5.8: Performance depending on the interval size for DDG hidden service for one request

5.4.2 Results

Obviously, the assumption used to calculate the theoretical result does not completely hold in a real application, as R is not an independent sequence because the cells from one request can fall in several intervals. In order to evaluate this assumption, we duplicate both hidden services, naming "Carol" to the second hidden service. We send requests to each hidden service from 10 different machines according to a Poisson model, as explained before, and Alice only makes requests to Bob but never to Carol. We use the Test (5.10) for Bob's flow, $\Lambda(s^L|H_1, x^n)$ and $\Lambda(s^L|H_0, x^n)$ for Carol's flow. For different values of η we obtain P_D as the rate of $\Lambda(s^L|H_1, x^n) > \eta$, and P_F as the rate of $\Lambda(s^L|H_0, x^n) > \eta$. We repeat the experiments 1000 times.

Figure 5.10 depicts the ROC for the DDG hidden service using $T = 0.5$, 1 and 1.5 s and Figure 5.11 depicts the ROC for SR hidden service using $T = 0.2$, 0.5 and

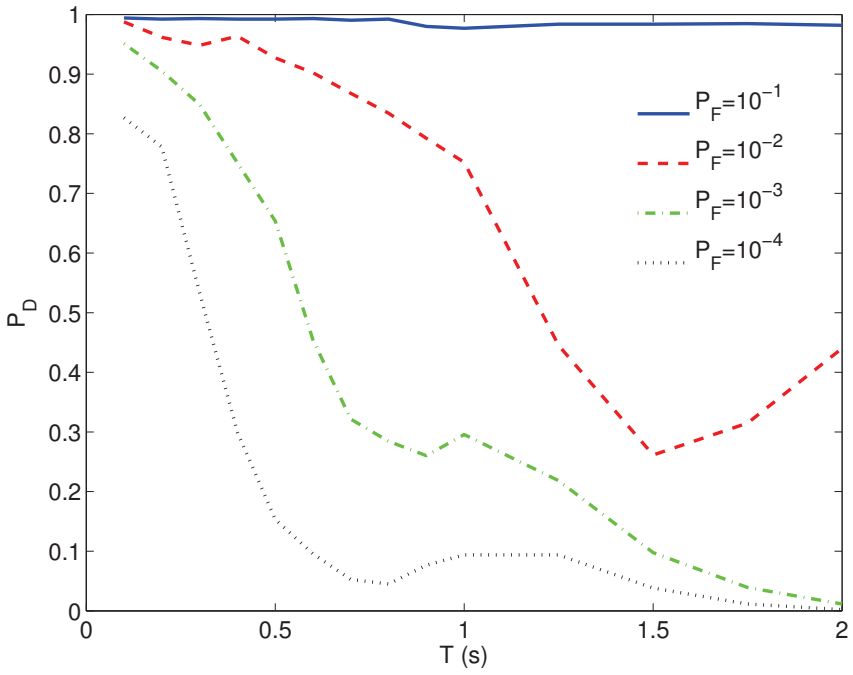


Figure 5.9: Performance depending on the interval size for SR hidden service for one request

1 s. We can see that in the real implementation the parameter T affects less than when R is chosen according to the model, i.e., an i.i.d. negative binomial sequence, and as we could expect, correlation between intervals decreases the performance.

5.4.3 Comparison with prediction-based technique

In this section, we compare the performance for this technique with the one developed in Chapter 4. For this purpose we use the the SR hidden service, but we increase the number of requests from other machines so that the hidden service receives requests according to a Poisson model at rate 6000 requests per hour. We make 2000 requests to the server and 1000000 different combinations of 10 requests. We decode using two methods: the one described in Chapter 4 and the one described previously in this chapter, depicting the results in Figure 5.12. We can see that for this scenario,

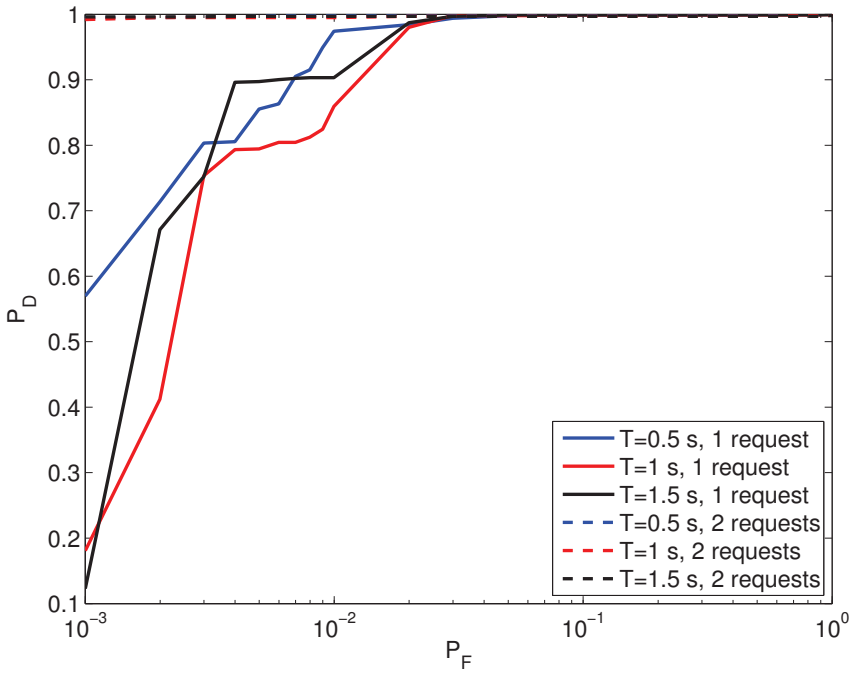


Figure 5.10: ROC for DDG hidden service

locating a Tor hidden service, the method presented in the previous chapter gives better results. This is due to using the whole timing information instead of just interval information, and even though the prediction-based method only uses the information of cells from client to the hidden service, I, discarding the information of B cells, those cells are the ones that carry more information for the test as they can be predicted more accurately.

5.5 Conclusions

This chapter presents a non-blind fingerprint for a flow created by the client. This method is based on counting the number of packets in certain intervals. To improve performance, we predict the time that the packet should be at the eavesdropping place and we use an optimal decoder.

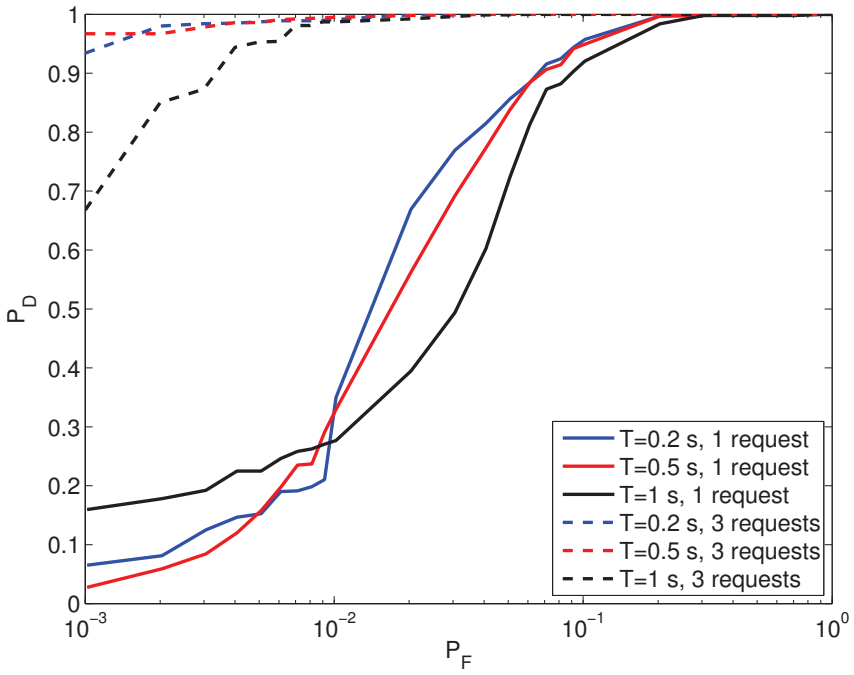


Figure 5.11: ROC for SR hidden service

The performance is studied empirically in the live Tor network to locate a hidden service. We also compare the results with the previous method, showing that performance is not as good as with the method presented in Chapter 4, but we also claim that under certain conditions, for instance where I and B flows are similar, this method could perform better than the pediction-based one.

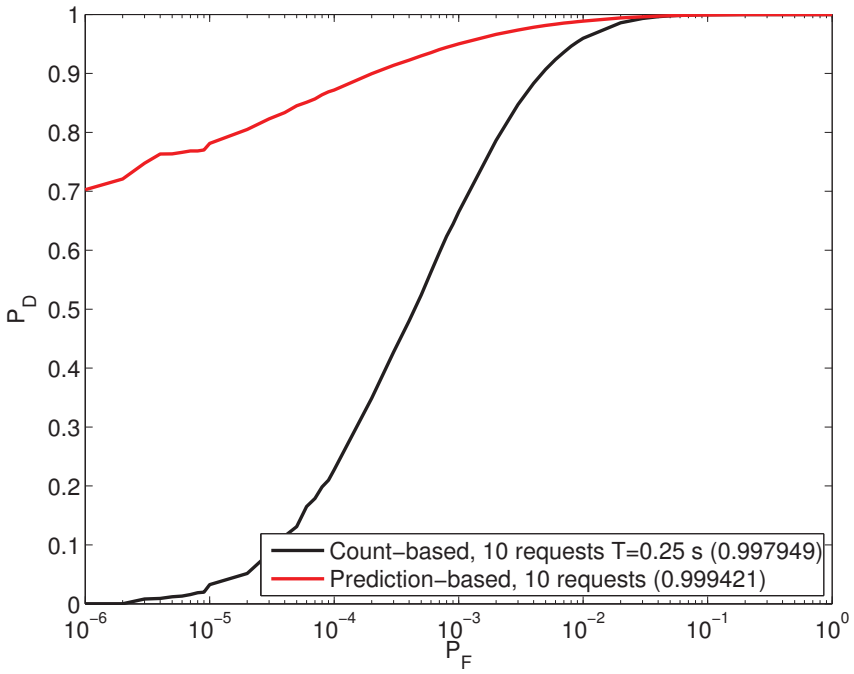


Figure 5.12: Performance for SR hidden service with 6000 requests per hour using 10 requests

Chapter 6

Inter-packet-delays-based Flow Correlation

6.1 Introduction

In this chapter we propose a second method to solve the problem set out in Chapter 2, i.e., deciding if the eavesdropped traffic is carrying a flow of which we have the timing information. Contrary to the approach of Chapter 4, this second method does not need an estimator of the time that packets arrive to the TA detector, hence this method can be applied even when the network in the middle is unknown.

The proposed method saves the Inter-packet delays (IPDs) of the flow and uses a detector based on the likelihood ratio test (Neyman-Pearson lemma). This method outperforms any of the state-of-the-art traffic watermarking schemes even using passive traffic analysis. For instance, 21 packets separated at least 10 ms are enough to correlate two flows, one in Virginia, the other in California, correctly with probability 0.9861 when the false positive probability is fixed to 10^{-5} and no countermeasures are exerted.

As IPDs are not robust against the insertion and drop of packets, we develop a modification which is robust against chaff packets, repacketization, flow splitting, and attacks that add or remove packets from the flow. We also make it robust against random delays under a maximum delay constraint.

6.2 Proposed Scheme

This section recalls the notation we use and explains how we correlate the flows to decide whether they are linked or not.

Figure 2.1 illustrates our system model. A flow of length n packets, that we are interested in tracking, goes through a certain link, termed "creator", where we can measure its packet timing information, X^n. The ith IPD at the creator is defined as $\Delta X_i = X_{i+1} - X_i,\ i = 1, \ldots n-1$, and these values are saved for later use in detection. This flow continues through the network without any modification.

The "detector" is another link in which we can measure the timing information, $Y^n = x^n + D^n$, where D_i is the network delay suffered by the ith packet. Then, the IPDs at the detector are

$$\Delta Y_i = Y_{i+1} - Y_i = X_{i+1} - X_i + D_{i+1} - D_i = \Delta X_i + \Delta D_i, \quad i = 1, \ldots, n-1 \quad (6.1)$$

where ΔD represents the PDV, also known as jitter. Note that we have assumed that there are no packets added or dropped to the flow, this assumption allow us to simplify the notation from Chapter 2 using Y_i instead of Y_{x_i}.

By using the information of the actual values ΔX^n and ΔY^n, the detector has to decide correctly if the two flows are linked. Two flows are linked if they follow a common timing pattern due to sharing the same source (i.e. the unencrypted payload is the same). Formally, we can express this problem via classical hypothesis testing

with the following hypotheses:

H_0: The flows are not linked.

H_1: The flows are linked.

6.3 Basic Detector

In this section we derive our detector and model the distributions of PDVs and IPDs as needed.

6.3.1 Detector construction

In order to obtain the best possible performance, we construct the optimal detector, which is the likelihood ratio test. Neyman-Pearson lemma proves that this test is the most efficient one between two simple hypotheses [83]. Hence, our detector chooses H_1 when

$$\Lambda(\Delta Y^n, \Delta X^n) = \frac{\mathcal{L}(H_1|\Delta Y^n, \Delta X^n)}{\mathcal{L}(H_0|\Delta Y^n, \Delta X^n)} = \frac{f(\Delta Y^n|\Delta X^n, H_1)}{f(\Delta Y^n|\Delta X^n, H_0)} > \eta. \tag{6.2}$$

and H_0 in the opposite case. $\mathcal{L}$ represents the likelihood function and η is a threshold that we fix to achieve a certain probability of false positive.

Recall from (6.1) that if H_1 holds, then $\Delta Y^n = \Delta X^n + J^{n-1}$. Conversely, if H_0 holds, ΔY^n is a sequence with joint pdf $f_{\Delta Y^n}(\Delta y^n)$.

For feasibility reasons, we constrain the detector to use first-order statistics, discarding the information carried by higher-order statistics. This is equivalent to assuming sample-wise independence in the sequences ΔD^n and ΔY^n.

In fact, independence in ΔD^n does not hold as both Δd_i and Δd_{i-1} depend on the value of the network delay d_i. We show this in Figure 6.1. The assumption of

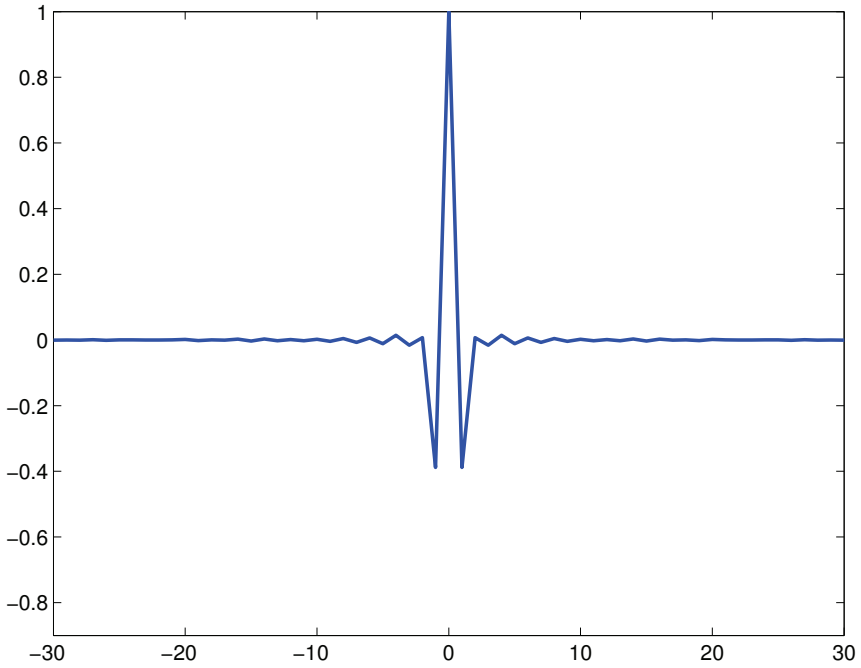


Figure 6.1: Autocorrelation of the PDV

independence in ΔY^n is more reasonable as shown in Figure 6.2. In Section 6.4.2 we quantify the impact of this assumption on performance comparing the real results with those that would be obtained for independent and identically distributed (i.i.d.) sequences.

Under these assumptions, the likelihood ratio becomes

$$\Lambda(\Delta Y^n, \Delta X^n) = \prod_{i=1}^{n-1} \frac{f_{\Delta D}(\Delta y_i - \Delta x_i)}{f_{\Delta Y}(\Delta y_i)}. \tag{6.3}$$

Therefore, we need to model the PDVs and the IPDs, i.e., determine $f_{\Delta D}(\Delta d)$ and $f_{\Delta Y}(\Delta y)$.

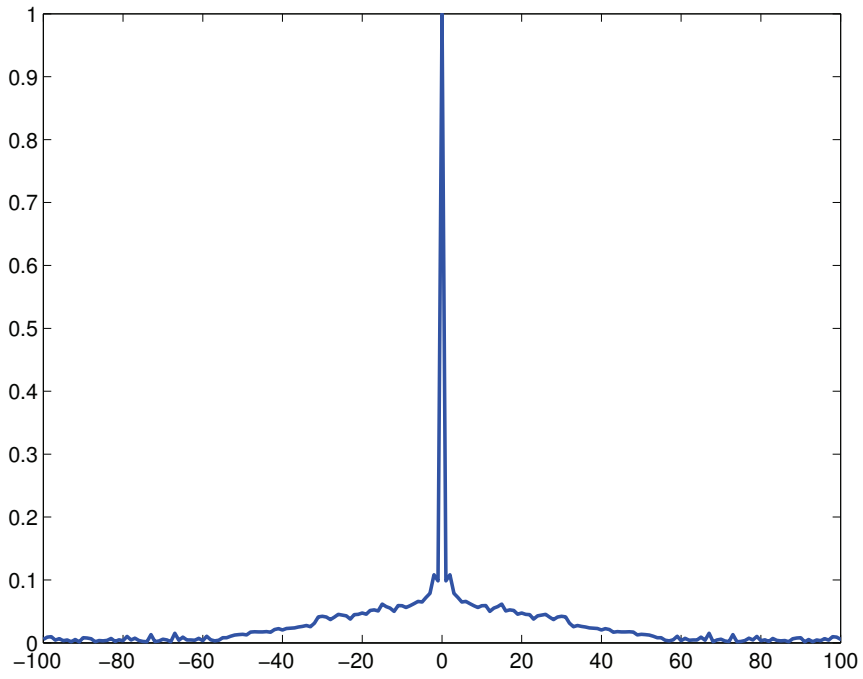


Figure 6.2: Autocorrelation of the IPD

6.3.2 Modeling the packet delay variation

To model the distribution of the PDVs, we first measure them in several real connections, then fit these data to some candidate distributions and select the distribution that matches best.

The measured delays are reported in [117]. This dataset contains the delays between two hosts during 72 hours, and for 11 different scenarios. As it is customary, we separate these data into three subsets: training, validation and test, using 24 hours of data for each.

Scenarios 1 to 9 measure common Internet connections between two hosts. Scenario 10 models the delays of a stepping-stone scenario, where a host in Oregon is retransmitting to a host in California the flow coming from a host in Virginia. Scenario 11 measures the delays associated with one instance of the Tor network [33].

	Source	Dest.	$\bar{n}$ [ms]	Var. [s^2]	P_{NL}
Sc1	CA-US	NM-US	15	$8 \cdot 10^{-5}$	$2 \cdot 10^{-4}$
Sc2	OR-US	NM-US	27	$1 \cdot 10^{-5}$	$6 \cdot 10^{-4}$
Sc3	VA-US	NM-US	41	$1 \cdot 10^{-3}$	$2 \cdot 10^{-3}$
Sc4	ES	NM-US	94	$7 \cdot 10^{-6}$	0
Sc5	IE	NM-US	74	$5 \cdot 10^{-5}$	$1 \cdot 10^{-4}$
Sc6	JP	NM-US	69	$6 \cdot 10^{-5}$	$1 \cdot 10^{-4}$
Sc7	AU	NM-US	109	$1 \cdot 10^{-4}$	$1 \cdot 10^{-3}$
Sc8	BR	NM-US	80	0.029	$1 \cdot 10^{-3}$
Sc9	SG	NM-US	110	$8 \cdot 10^{-5}$	$1 \cdot 10^{-1}$
Sc10	VA-US	CA-US	63	0.006	$5 \cdot 10^{-3}$
Sc11	NM-US	NM-US	3117	0.265	0.16

Table 6.1: Basic Statistics of the measured delays

In order to get a general idea about the connection scenarios, we show some basic information of the hosts and the connections in Table 6.1, where P_{NL} is the probability of packet loss and the source and destination are represented with ISO 3166 codes [118].

From these measured delays we calculate the measured PDV as $j_i = d_{i+1} - d_i$. The basic statistics from Table 6.2 imply a nearly symmetric (i.e., small skewness) and leptokurtotic distribution (i.e., sharp peak and heavy tail).

To construct the model, we make the same assumptions as to build the test, i.e., an i.i.d. sequence. The candidate distributions were selected among the ones that have support on $\mathbb{R}$ and possess the mentioned characteristics. The chosen distributions are Cauchy, Gumbel, Laplace, Logistic and Normal. Their pdfs are summarized in Table 6.3. Recall that $\mathbb{1}_{[a,b]}(x)$ is the indicator function that takes the value 1 when $x \in [a, b]$, and is 0 otherwise.

We estimate the respective parameters using robust statistics, to prevent that outliers affect the measurements. These estimators are based on the median and median absolute deviation and calculated as explained in [116, Chapter 3]. Afterwards, we measure the goodness of fit between the validation sequence and the model using

	$\bar{\Delta}d$ [s]	Var. [s^2]	Skew.	Kurtosis
Sc1	$1 \cdot 10^{-10}$	$1 \cdot 10^{-4}$	0.02	83185
Sc2	$-2 \cdot 10^{-10}$	$1 \cdot 10^{-5}$	5.84	408
Sc3	$-2 \cdot 10^{-10}$	$2 \cdot 10^{-3}$	0.003	85187
Sc4	$1 \cdot 10^{-9}$	$1 \cdot 10^{-6}$	17.1	81139
Sc5	$-7 \cdot 10^{-9}$	$3 \cdot 10^{-6}$	3.81	622
Sc6	$-2 \cdot 10^{-9}$	$6 \cdot 10^{-5}$	0.78	71212
Sc7	$2 \cdot 10^{-8}$	$2 \cdot 10^{-5}$	-0.01	78821
Sc8	$9 \cdot 10^{-9}$	$6 \cdot 10^{-3}$	-10^{-5}	19893
Sc9	$2 \cdot 10^{-8}$	$4 \cdot 10^{-6}$	4.41	620
Sc10	$-8 \cdot 10^{-9}$	$3 \cdot 10^{-4}$	2.37	22789
Sc11	$-1 \cdot 10^{-6}$	$6 \cdot 10^{-3}$	2.97	410

Table 6.2: Basic Statistics of the measured PDV.

the square root of the JSD (Jensen-Shannon Divergence), D_{JS} [82]. Recall from Chapter2, that JSD is a metric for two probability densities P, Q, which is based on the KLD as follows:

$$D_{JS}(P,Q) = \sqrt{\frac{1}{2}\left(D(P \mid\mid M) + D(Q \mid\mid M)\right)} \tag{6.4}$$

where $M = \frac{1}{2}(P+Q)$ is the mid-point measure, and $D(\cdot||\cdot)$ is the KLD.

Distrib.	**pdf**
Cauchy	$f(x\|\mu,\sigma) = \frac{1}{\pi\sigma\left(1+(\frac{x-\mu}{\sigma})^2\right)}$
Gumbel	$f(x\|\mu,\sigma) = \frac{1}{\sigma} exp\left(-\frac{x-\mu}{\sigma} - exp\left(-\frac{x-\mu}{\sigma}\right)\right)$
Laplace	$f(x\|\mu,\sigma) = \frac{1}{2\sigma}\exp -\frac{\lvert x-\mu\rvert}{\sigma}$
Logistic	$f(x\|\mu,\sigma) = \frac{\exp\left(-\frac{x-\mu}{\sigma}\right)}{\sigma\left(1+exp\left(-\frac{x-\mu}{\sigma}\right)\right)^2}$
Normal	$f(x\|\mu,\sigma) = \frac{1}{\sigma\sqrt{2\pi}} exp\left(-\frac{(x-\mu)^2}{2\sigma^2}\right)$
Exp.	$f(x\|\lambda) = \lambda\exp(-\lambda x)\mathbb{1}_{[0,\infty)}(x)$
Pareto	$f(x\|\alpha, x_m) = \frac{\alpha x_m^\alpha}{x^{\alpha+1}}\mathbb{1}_{[x_m,\infty)}(x)$
LogNor.	$f(x\|\mu,\sigma^2) = \frac{1}{x\sqrt{2\pi\sigma^2}} exp\left(-\frac{(\log x-\mu)^2}{2\sigma^2}\right)$
LogLog.	$f(x\|\alpha,\beta) = \frac{(\beta/\alpha)(x/\alpha)^{\beta-1}}{\left(1+(x/\alpha)^\beta\right)^2}\mathbb{1}_{[0,\infty)}(x)$
Weibull	$f(x\|\gamma,\beta) = \frac{\gamma}{\beta}x^{\gamma-1}\exp\left(-\frac{x^\gamma}{\beta}\right)\mathbb{1}_{[0,\infty)}(x)$

Table 6.3: Pdfs of the candidate distributions for PDV and IPD.

Scenario	Cau.	Gum.	Lap.	Log.	Nor.
Sc. 1	0.168	0.218	0.101	0.123	0.159
Sc. 2	0.156	0.201	0.157	0.150	0.171
Sc. 3	0.135	0.211	0.163	0.167	0.192
Sc. 4	0.294	0.369	0.252	0.270	0.296
Sc. 5	0.153	0.193	0.139	0.135	0.159
Sc. 6	0.203	0.174	0.152	0.120	0.130
Sc. 7	0.136	0.300	0.231	0.267	0.298
Sc. 8	0.168	0.307	0.195	0.261	0.308
Sc. 9	0.183	0.185	0.171	0.141	0.146
Sc. 10	0.227	0.384	0.340	0.364	0.384
Sc. 11	0.251	0.201	0.194	0.228	0.253

Table 6.4: Goodness of fit of the candidate distributions for PDV.

Results from Table 6.4 show that no distribution stands out above the rest, being the Laplace and the Cauchy distributions the best fits.

The Laplacian is the most commonly used model for the jitter, but an alpha-stable distribution models it better [119]. Note that a Cauchy distribution is a particular case of an alpha-stable distribution, but we do not generalize it further, as we are interested in a close-form pdf model.

The performance of the two possible detectors, based on Laplace and Cauchy distributions, respectively, is evaluated in Section 6.4.2.

6.3.3 Modeling the Inter-Packet Delays

In many works it is assumed a Poisson model for the traffic because of its desirable theoretical properties [120]. This model implies that IPD times are an i.i.d. exponentially-distributed sequence, but Paxson et al. [115] have shown that this model is not accurate in interactive applications.

We model the IPDs on both SSH and HTTP protocols. As done in [115], we only take into account packets that are separated at least by 10 ms, considering that if

Set	Flows	Packets
SSH Train.	6447	14442323
SSH Val.	1128	2594550
SSH Sim.	714	16595655
HTTP Train.	1108909	356620487
HTTP Val.	208896	63982082
HTTP Sim.	1007545	322853437

Table 6.5: Characteristics of the IPD sets.

Distribution	SSH		HTTP	
	Error	Parameters	Error	Parameters
Exponential	0.756	$\lambda = 5.46$	0.758	$\lambda = 12.69$
Pareto	0.149	$\alpha = 0.86,\ x_m = 10^{-2}$	0.247	$\alpha = 0.53,\ x_m = 10^{-2}$
Log-Normal	0.627	$\mu = -1.14,\ \sigma^2 = 1.43$	0.723	$\mu = -0.40,\ \sigma^2 = 4.02$
Log-Logistic	0.343	$\alpha = 0.27,\ \beta = 1.77$	0.508	$\alpha = 0.47,\ \beta = 0.95$
Weibull	0.554	$\gamma = 0.49,\ \beta = 0.81$	0.591	$\gamma = 0.40,\ \beta = 1.33$

Table 6.6: MLE Estimator and goodness of fit of the candidate distributions for IPD.

two packets are separated by less than 10 ms they are subpackets of the same packet. Therefore, the considered IPDs are lower bounded by 10 ms. We use the captures from Dartmouth College [121], using the traces from Fall 03 as training set, Spring 02 as validation set and Fall 01 as test set for the simulator. The basic characteristics of these sets are shown in Table 6.5.

We estimate the parameters through MLE and measure the goodness of fit using the square root of the JSD. The candidate distributions are: Exponential, Pareto, Log-Normal, Log-Logistic, and Weibull. Their pdfs can be seen in Table 6.3.

Results shown in Table 6.6 confirm the findings of Paxson et al., i.e., that the Pareto distribution is a better model for interactive traffic. In non-interactive traffic such as HTTP, this model also gives acceptable results. Therefore, we will assume that

$$f_{\Delta Y}(\Delta y) = \alpha x_m^{\alpha} \Delta y^{-\alpha-1} \mathbb{1}_{[x_m,\infty)}(\Delta y). \tag{6.5}$$

6.3.4 Detector

Once we have a model for the IPD and PDV sequences, we derive the likelihood test.

If Cauchy-distributed PDVs are assumed, the test chooses H_1 when

$$\Lambda(\Delta y^n, \Delta x^n) = \prod_{i=1}^{n-1} \frac{(\Delta y_i)^{\alpha+1}}{\pi\sigma\alpha x_m^{\alpha}\left(1+\left(\frac{\Delta y_i - \Delta x_i}{\sigma}\right)^2\right)} > \eta \tag{6.6}$$

and H_0 otherwise.

In the case that a Laplace model for PDV is adopted, then

$$\Lambda(\Delta y^n, \Delta x^n) = \exp\left(-\frac{\sum_{i=1}^{n-1}|(\Delta y_i - \Delta x_i|}{\sigma}\right)\frac{\left(\prod_{i=1}^{n-1}\Delta y_i\right)^{\alpha+1}}{(2\sigma\alpha x_m^{\alpha})^{n-1}}. \tag{6.7}$$

When it comes to finding η and obtaining the theoretical probabilities of detection and false positives, we use the Monte Carlo method as the derivation of closed-form expressions is infeasible in most cases, as shown in Appendix A, where we obtain the theoretical η and P_D for tests (6.6) and (6.7).

6.4 Performance

In this section we construct a simulator and present the scenarios we use in the remaining of the paper. Afterwards, we test the model assumptions and measure the performance with different sequence lengths.

6.4.1 Simulator and Scenarios

.

Simulations are carried out in the following way. First, we generate timing information at the creator using the IPD test set, X_1^n. The purpose of this sequence is to

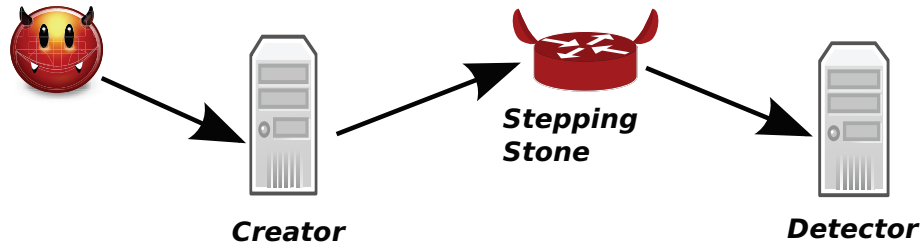


Figure 6.3: Simulated Scenario A.

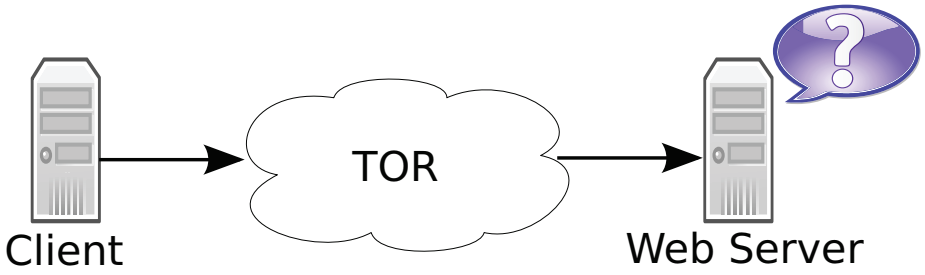


Figure 6.4: Simulated Scenario B.

evaluate the performance when H_1 holds. A delay is added to each packet using the measured delays from the test set (as explained in the following paragraphs), obtaining Y_1^n. We generate a second sequence Y_0^n, using the IPD test set; this sequence has the purpose of evaluating the performance under H_0. Finally, we use the Test from (6.6) or (6.7), to obtain both $\Lambda(d_0^n, c_1^n)$ and $\Lambda(d_1^n, c_1^n)$. This experiment is repeated 10^6 times, and for different values of η we obtain P_D as the rate of $\Lambda(d_1^n, c_1^n) > \eta$, and P_F as the rate of $\Lambda(d_0^n, c_1^n) > \eta$. Note that due to the number of runs, $P_F < 10^{-5}$ cannot be measured and results of this order are not accurate.

The sequences are generated in the following way: we place all the IPDs from the test set in an order-preserving list. The starting point is randomly selected from the list and the generated IPDs are the following $n-1$ values.

For generating the delays, we used the test set as a list with the delay every 50 ms. We select one value randomly from the list that will be considered time 0 ms; the following values will represent the delay at times 50 ms, 100 ms, and so on. To obtain

the delays at times where we do not have a measure, we use linear interpolation.

The performance is evaluated in the two scenarios depicted in Figures 6.3 and 6.4. Scenario A represents a stepping stone that forwards SSH traffic inside the Amazon Web Services [122] network. The creator, stepping stone and detector are EC2 instances located in Virginia, Oregon and California, respectively. This example corresponds to tracing the source of an attack that was launched from a compromised Amazon instance. The simulated delays correspond to those of Scenario 10 in Section 6.3.2, where the standard deviation of the network delay is 4 ms.

Scenario B simulates a web page accessed from Tor network whose real origin is to be found, and where the creator will be the web page and the detector the client. For instance, this case can correspond to a company in whose forum an anonymous insulting post has been placed using Tor and it is to be known whether the source comes from an employee within the company. The simulated delays correspond to the measurements of Scenario 11 in Table 6.4, where the standard deviation of the network delay is 340 ms.

6.4.2 Impact of our assumptions

In this section, we wish to quantify the impact of the assumptions we have made, that is, the PDVs form an i.i.d. Cauchy or Laplace sequence. To this end, we extend our simulator to create 3 types of delays: first, according to the model (Cauchy or Laplace), second as a random sample from the data, and last, from the data maintaining the time correlation. $n = 4$ is used for Scenario A and $n = 21$ for Scenario B. Results are shown in Figures 6.5 and 6.6. We notice two details: first, that the Cauchy-based detector gives slightly better performance than the Laplace under real data, and second, that the independence of the PDVs previously assumed slightly reduces the performance. In the sequel, we just derive the expressions for a Cauchy-based detector. The modification for a Laplace detector is rather straightforward.

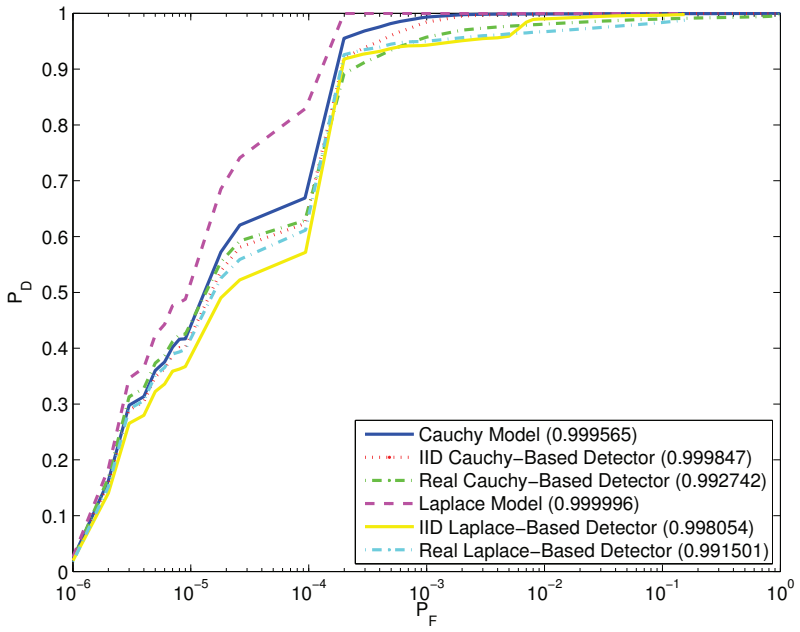


Figure 6.5: Impact of assumptions in Scenario A with $n = 4$.

6.4.3 Performance dependence on n

We want to evaluate how much performance is improved when longer sequences are used. The result is depicted in Figures 6.7 and 6.8. We can see that Scenario B, whose IPDs have a larger variance because of the Tor network, needs much longer sequences to achieve the same performance. For instance, with fixed $P_F = 10^{-4}$, in Scenario A for $n = 6$ we obtain $P_D = 0.8926$. However, in Scenario B the n needed for a comparable result is around 250, with which we obtain $P_D = 0.8947$. If we compare the AUCs (Area Under the ROC Curve), in Scenario A with $n = 6$ we obtain 0.9955 while a similar result in Scenario B requires a value of n between 10 and 25.

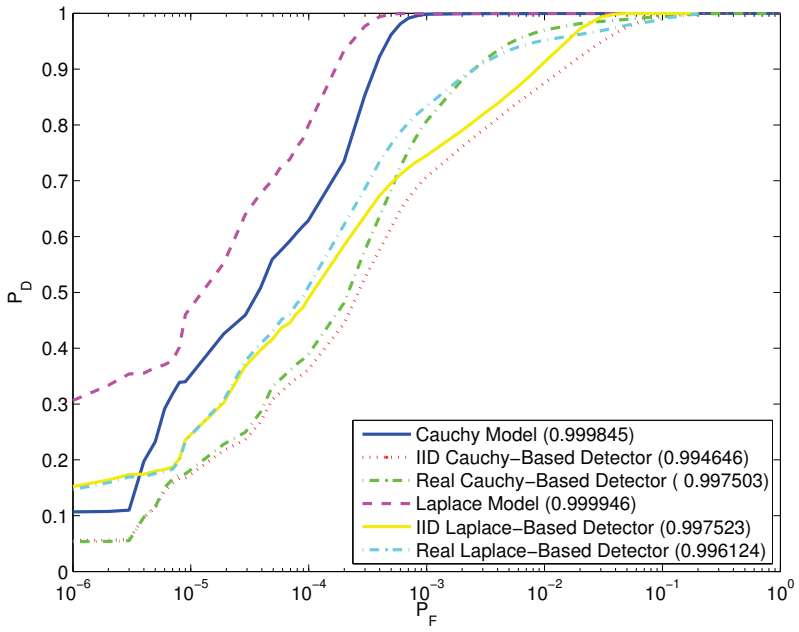


Figure 6.6: Impact of assumptions in Scenario B with $n = 21$.

6.5 Robust detector

The previous test does not take the existence of any countermeasure into account. Attacks to timing correlation can be exerted by introducing uncorrelated random delays, adding chaff traffic or splitting the flow, making the Test in (6.6) ineffective. In this section, we build a test that is robust to these attacks. First, we deal with adding or removing packets from the flow, and then with random delays.

6.5.1 Matching packets

Hitherto, we have assumed that there is a one-to-one relation between the flows at the creator and the detector; i.e., no packets are added or removed. This assumption is not necessarily valid for every situation, not only due to the presence of an active attacker, but also as a result of many applications that repacketize flows, changing

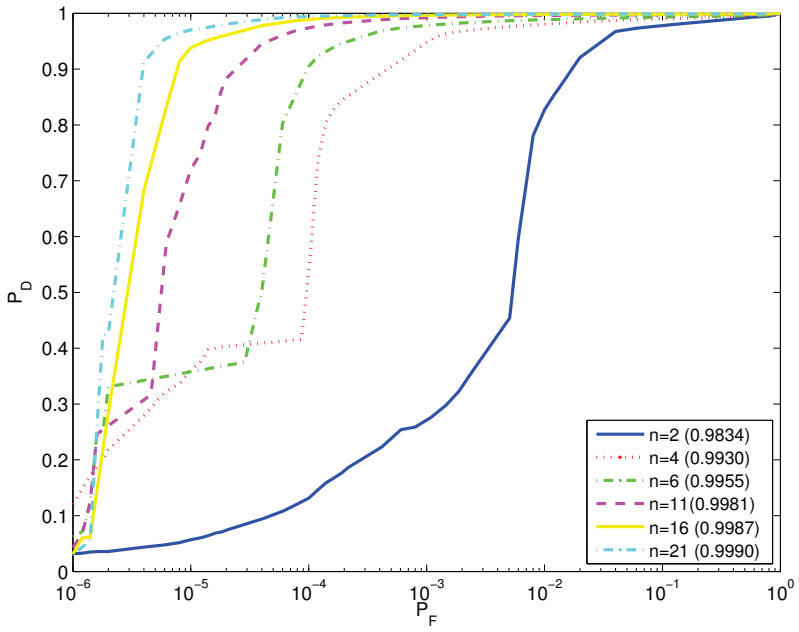


Figure 6.7: Performance dependence with n in Scenario A.

the number of packets, for instance, SSH tunneling [123].

To deal with packet addition and removal, we first choose the most likely packet at the detector for each packet at the creator. In the case that there is no packet likely enough, we consider the creator packet as lost.

Given the ith packet at the creator, we match it with the most likely jth packet at the detector, denoting this as $i \rightarrow j$. Consequently, if ρ is a synchronization constant to be discussed in Section 6.5.3, and γ is the threshold for which a packet is considered lost, the condition for a match in the ith packet is

$$|x_i - (y_j - \rho)| < |x_i - (y_k - \rho)|, \quad \forall k \neq j, \tag{6.8}$$

and to avoid considering it lost,

$$|x_i - (y_j - \rho)| < \gamma. \tag{6.9}$$

Threshold γ should be large enough so that the probability P_M that a packet is

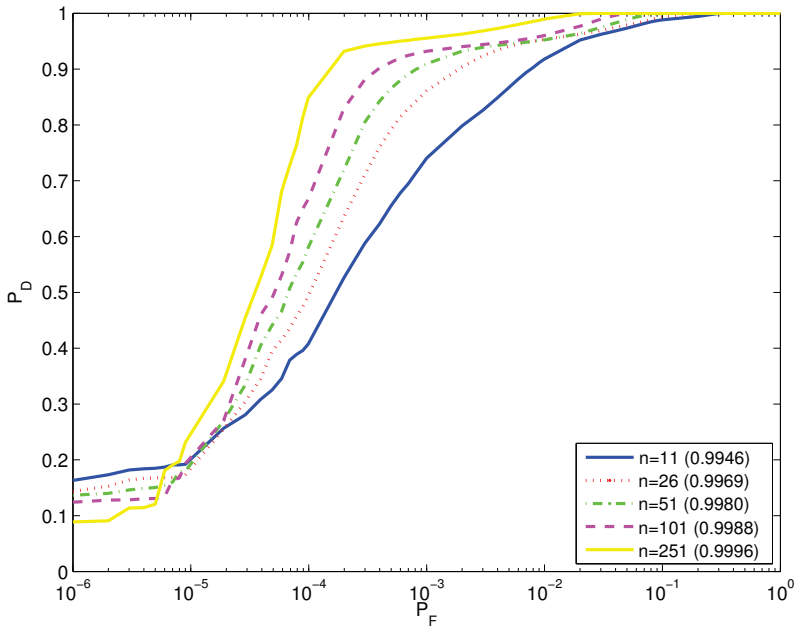


Figure 6.8: Performance dependence with n in Scenario B.

wrongly considered lost is very small, for instance, 10^{-6}. Although this can lead to incorrectly matching with another packet when the packet is indeed lost, the impact on Test (6.10) of this mismatch is very small. Empirically, the best performance we obtained for Scenario A is when $\gamma \approx 75$ ms, and when $\gamma \approx 7$ s for Scenario B.

In practice, the standard deviation of the network delay can be larger than some of the IPDs, especially in Scenario 2, in which case the matching is likely to fail. The impact of these matching errors is evaluated in Section 6.5.5. In the case that most of the IPDs are smaller than the standard deviation of the network delay, a better matching function is the one proposed in Chapter 7. This corresponds to the injective function that minimizes the mean square error between x^n and $y^m - \rho$, which has the drawback of a higher computational cost.

The matching process modifies the timing sequences to x^m and y^m, where $m \leq n$, as the lost packets are removed. Formally, we can define the new sequences as $x^m = \{x_i \mid \exists\, j :\ i \to j\}$, and $y^m = \{y_j \mid \exists\, i :\ i \to j\}$.

6.5.2 Test robust to chaff and flow splitting

From (6.6), we can obtain a test robust to packet removal and insertion as

$$\Lambda(\Delta x^m, \Delta y^m) = {P_L}^{n-m} \cdot \prod_{i=1}^{m-1} \left(P_L + (1-P_L) \cdot \frac{(\Delta y_i)^{\alpha+1}}{\pi\sigma\alpha x_m^\alpha \left(1 + \left(\frac{\Delta y_i - \Delta x_i}{\sigma}\right)^2\right)} \right), \quad (6.10)$$

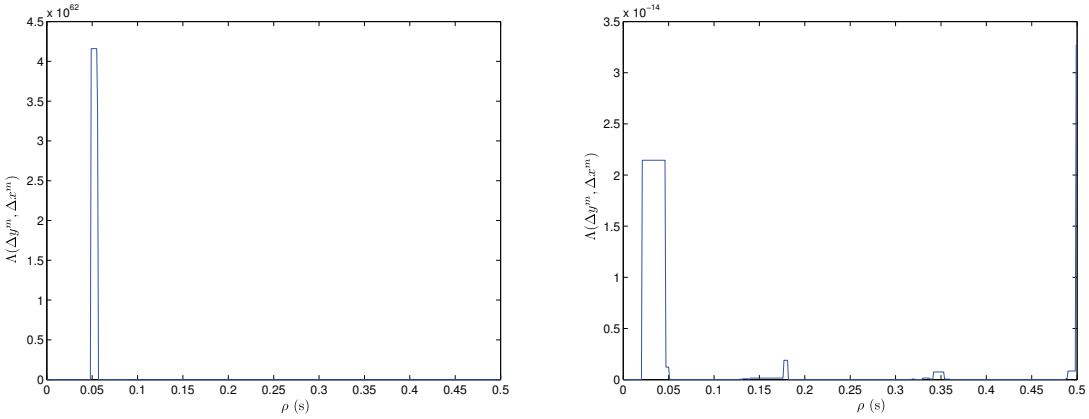
where P_L is the probability that a packet at the creator cannot be matched at the detector. This can be due to three reasons: network loss with a probability P_{NL}, lack of matching when the packet appears, and flow splitting into S subflows by the stepping stone, i.e., $1/S$ of the original packets are seen by the detector, as only one of the subflows traverses this link. Therefore,

$$\begin{aligned} P_L &= 1 - \frac{(1-P_{NL})(1-P_M)}{S} = \frac{S-1+P_{NL}}{S} + \frac{P_M - P_{NL}P_M}{S} \\ &\approx \frac{S-1+P_{NL}+P_M}{S} \end{aligned} \quad (6.11)$$

6.5.3 Self-Synchronization

We have mentioned that ρ is a synchronization constant. The detector can perform detection maximizing the value of $\Lambda(\Delta x^m, \Delta y^m)$ with respect to ρ through an exhaustive search. For instance, Figure 6.9 shows a detector trying values of ρ using steps of 1 ms in the interval $[0, 0.5]$ s. We can see that the maximum $\Lambda(\Delta x^m, \Delta y^m)$ occurs when $\rho \approx \bar{n}$, as expected. Recall that $\bar{n}$ is the sample mean of the network delays.

In a real situation, calculating P_L with (6.11) may not be feasible due to S being unknown for the detector, and P_{NL} can be difficult to estimate as packets may go through several stepping stones. In that case, we propose to use also exhaustive search. Figure 6.10 shows a detector trying different values for P_L in steps of 0.01. We can see that the maximum $\Lambda(\Delta x^m, \Delta y^m)$ occurs at a value in a neighborhood of the real $P_L = 0.75$. We can also see that the peak of $\Lambda(\Delta x^m, \Delta y^m)$ is fairly wide, so not such small step is needed for the search in practice.

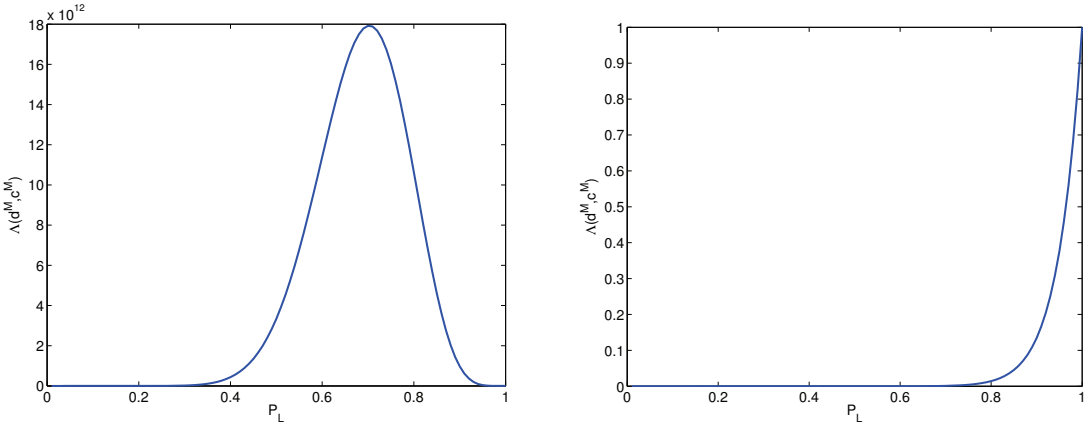


(a) Linked flows (b) Non linked flows

Figure 6.9: Synchronization in Scenario A with $n = 6$.

6.5.4 Robust test against random delays

So far the situation where an attacker can inject random delays has not been considered. Random delay injection is a well-known technique for covert channel prevention and can be easily implemented via buffering by attackers across their step stones.



(a) Linked flows (b) Non linked flows

Figure 6.10: Exhaustive search for P_L in Scenario A with $n = 21$ and real value of $P_L = 0.75$.

We assume that the attacker has the constraint of not being able to delay any packet more than A_{max} seconds. Hence, she can modify the PDV by a quantity A that falls in the interval $[-A_{max}, A_{max}]$. As we do not know the distribution of the attacker's random delay, the detector assumes a uniform distribution. Thus, the PDV at the decoder is $\Delta D'^m = \Delta D^m + A^m$, and

$$f_{\Delta D'}(j) = \frac{1}{2A_{max}\pi}\left((\arctan\left(\frac{j + A_{max}}{\sigma}\right) - \arctan\left(\frac{j - A_{max}}{\sigma}\right)\right) \tag{6.12}$$

Consequently, the likelihood ratio becomes

$$\Lambda(\Delta y^m, \Delta x^m) = {P_L}^{n-m} \cdot \prod_{i=1}^{m-1}\left(P_L + (1 - P_L) \cdot \frac{(\Delta y_i)^{\alpha+1} f_{\Delta D'}(\Delta y_i - \Delta x_i)}{\alpha x_m^{\alpha}}\right) \tag{6.13}$$

A game-theoretic approach to this problem is taken in Chapter 7, where the detector is first constrained to estimating and compensating the attack and then the optimal detector for the same game is derived. Results show that a nearly deterministic attack impairs the detector performance more than a uniform distribution even if the detector knows the attack distribution.

6.5.5 Performance

To evaluate the proposed robust algorithms, the functionalities of adding chaff traffic, splitting the flow, and delaying the packets randomly are implemented in our simulator. This is done as follows: each packet is delayed by a certain quantity. We implement two different delay strategies: a) the value is picked from a uniform distribution in the range $[0, A_{max}]$, and b) the values are taken to minimize (6.10), i.e. the values are chosen by an intelligent adversary who knows both the test and its parameters. Then, the simulator adds traffic according to a Poisson process with a fixed rate proportional to the rate of the original traffic. Afterwards, it simulates the flow split, which is implemented by discarding packets as a Bernoulli process with a probability equal to $1 - \frac{1}{S}$. Recall that S is the number of subflows we divide the flow into.

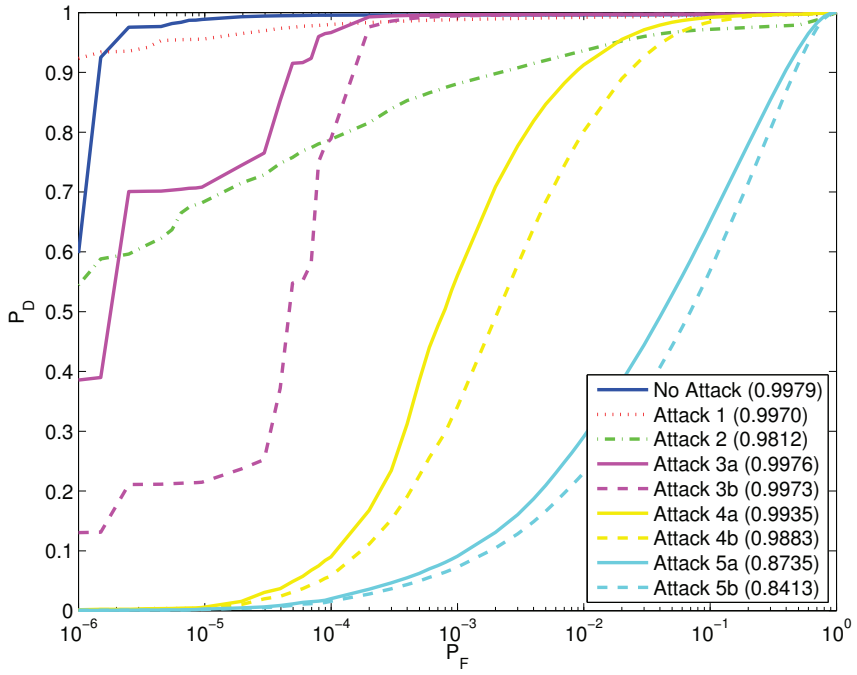


Figure 6.11: Performance under different traffic modifications in Scenario A, n=21.

We created five different attacks. In the first three, we evaluate each traffic modification strategy separately, namely, Attack 1 adds 500% of chaff traffic; Attack 2 splits the flow into 4 subflows; Attack 3 adds delays with $A_{max} = 50$ ms; Attack 4 combines 500% of chaff traffic with delays constrained to $A_{max} = 50$ ms, and Attack 5 is a complex attack where a combination of Attack 4 with splitting the flow into 2 subflows takes place. For Attacks 3 to 5, we consider the two delay strategies specified above: with Z indicating the attack number, we denote by Za the case where the delays are chosen randomly, and by Zb where they are chosen by an intelligent attacker. We simulate these situations using sequences of length $n = 21$ in Scenario A and $n = 251$ in Scenario B. Results are depicted in Figures 6.11 and 6.12.

Comparing these figures under no attacks with the corresponding plots for the case of no mismatches of Figs. 6.7 and 6.8, we can evaluate the impact of mismatched

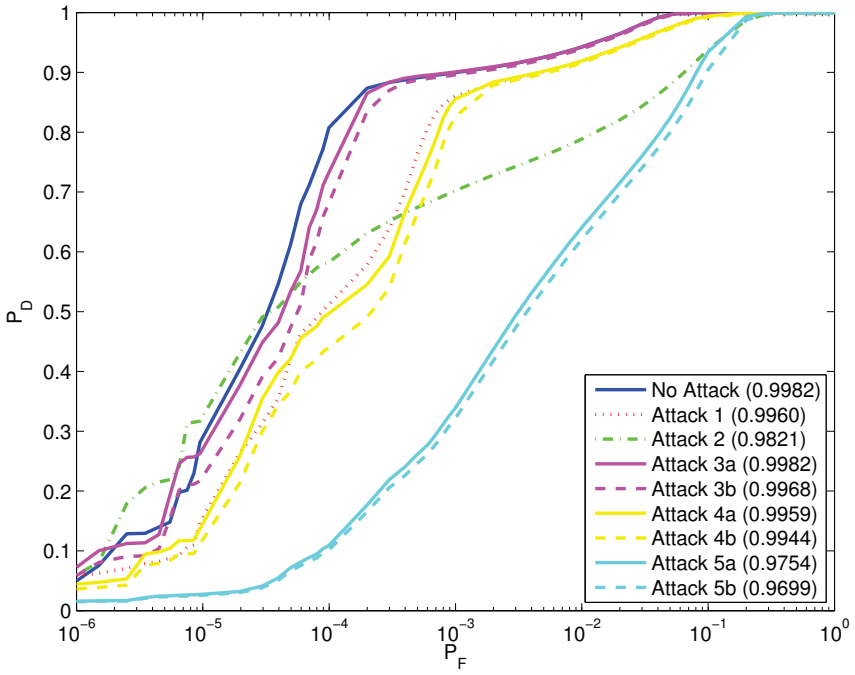


Figure 6.12: Performance under different traffic modifications in Scenario B, n=251.

packets, as the AUC drops from 0.9990 to 0.9979 in Scenario 1 and from 0.9996 to 0.9982 in Scenario 2.

In low jitter situations, namely Scenario A, chaff traffic by itself has little impact, but the effect when combined with random delays is significantly increased. The reason behind this is that in the first case the matching process chooses the real packets with a very low probability of error but when a random delay is added the probability of a mismatch increases. We also see that the flow splitting attack has a considerable impact as the received sequence length is reduced.

In high jitter situations, i.e. Scenario B, random delays have considerably smaller influence, because the standard deviation of the network delay is larger than the attack delay. In fact, due to the high network-delay variability, chaff traffic alone has a significant impact on performance without the need of an attacker injecting random delays.

6.6 Comparison with an active watermark

We want to analyze how much performance can be improved by sacrificing undetectability. For this purpose, we create an active watermark designed with invisibility as a goal, and we study the trade off between performance and detectability.

We measure the latter as the KLD between the covertext, i.e., the sequence without watermark, and the stegotext, i.e., watermarked. Cachin [103] defines a stegosystem to be ϵ-secure against passive adversaries if $D(f_C||f_S) < \epsilon$, where f_C is the distribution of the covertext and f_S is the distribution of the stegotext. Hence, we measure the detectability as the minimum ϵ for which our system is ϵ-secure.

The watermark is embedded adding a random uniform delay between $[0, W_{max}]$. Thus, the watermarked flow is $x^n = u^n + w^n$, hence ΔW^n is triangular-distributed between $[-W_{max}, W_{max}]$ as it is the difference of two delays uniformly distributed. As the saved IPDs are the ones after embedding the watermark, the detector remains (6.13).

We assume that the attacker knows the original traffic as done in [102, 99] and wants to test for the existence of a watermark. Therefore, the attacker's goal is to differentiate between $\Delta(W^n + D^n)$ and ΔD^n.

We simulate Scenario A with $n = 6$ and Scenario B with $n = 26$ under no traffic modification, where we evaluate the trade-off between the detectability and P_D when P_F is fixed. Results are depicted in Figures 6.13 and 6.14, where we can see that watermarking schemes give a significant improvement under low-jitter conditions even with $W_{max} = 2$ ms, (where $D_{KL}(\Delta D||\Delta W^n + \Delta D^n) = 0.486$), but this improvement is significantly lower on large-jitter conditions, e.g. the Tor network, even of very large W_{max} , for instance, for $W_{max} = 250$ ms (where $D_{KL}(\Delta D||\Delta W^n + \Delta D^n) = 0.679$).

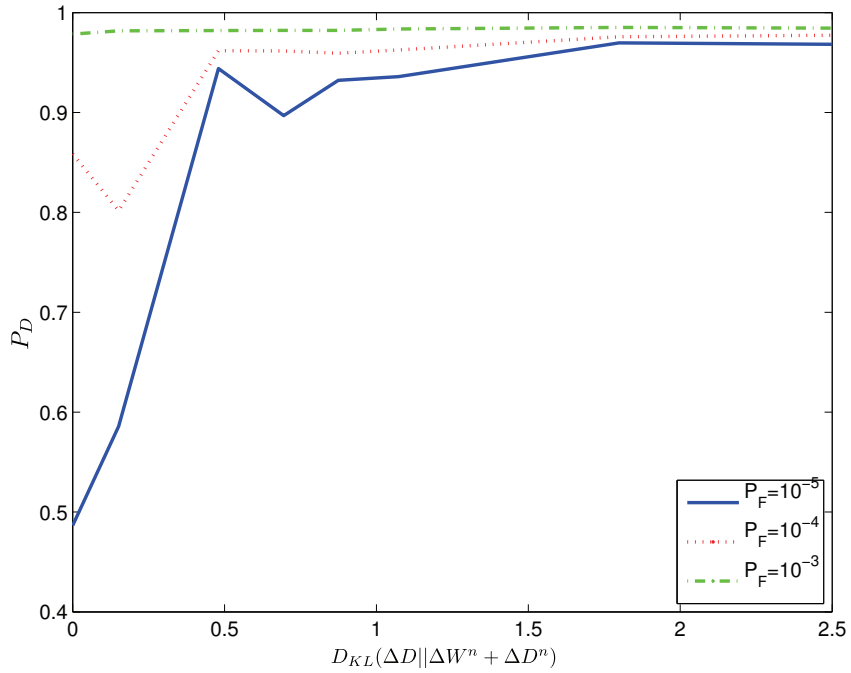


Figure 6.13: P_D vs the detectability for fixed P_F in Scenario A with $n = 6$.

6.7 Comparison with other schemes

We want to compare our passive analysis with four other state-of-the-art traffic watermarking schemes: IBW (Interval Based Watermark) [90], ICBW (Interval-Centroid-Based Watermark) [91] , RAINBOW (Robust And Invisible Non-Blind Watermark) [88] and SWIRL (Scalable Watermark that is Invisible and Resilient to packet Losses) [89] that where described in Chapter 2. To this end, we extend our simulator to be able to embed the mentioned watermarks and to detect them.

The presented results have been obtained with the following parameters: IBW, ICBW and SWIRL use a time interval of 500 ms; this is the value used in the original ICBW experiments reported in [91].The experiments for SWIRL in [89] use 2 s, but with short sequences this implies that many flows cannot be watermarked as the whole flow falls into one interval. We compensate this shorter interval by

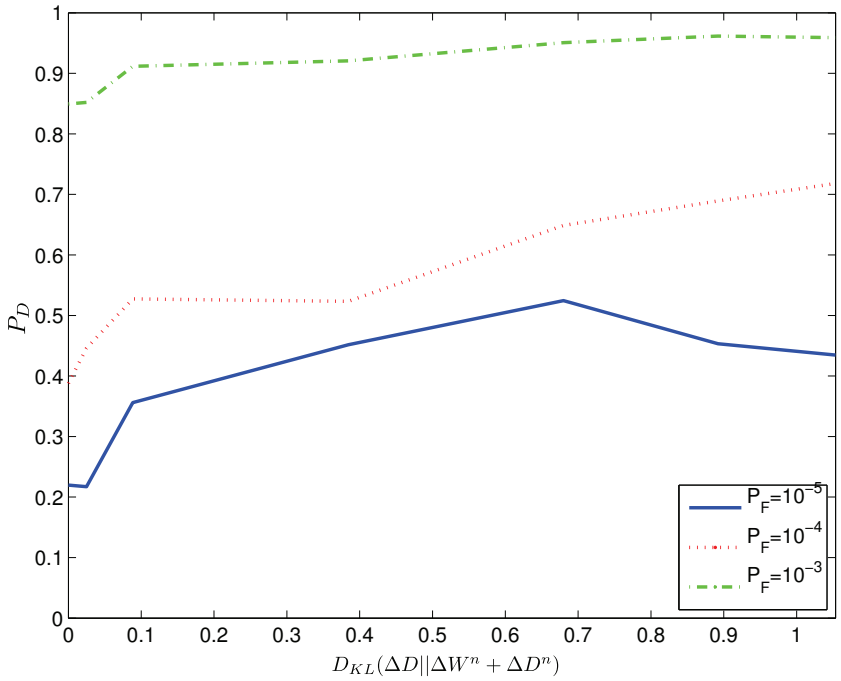


Figure 6.14: P_D vs the detectability for fixed P_F in Scenario B with $n = 26$.

dividing it into less subintervals (5 instead of 20). In our experiments RAINBOW can modify the IPD up to 20 ms, which is the largest watermark amplitude used in the simulations in [88].

We first compare the performance in both scenarios when the flows do not suffer any addition or removal of packets, for this we use (6.6). We take $n = 6$ in Scenario A and $n = 51$ in Scenario B. Figure 6.15 shows the results for Scenario A, where our scheme and RAINBOW outperform the rest by a significant amount. This is due to the fact that both are non-blind and perform well with short sequences if the PDV has small variance. The other watermarking schemes do not perform well with short sequences. Figure 6.16 shows the results in Scenario B. We see that with longer sequences IBW and ICBW despite of the larger PDV sequence improve their performance.

We also compare the performance under traffic modification using Attack 5a,

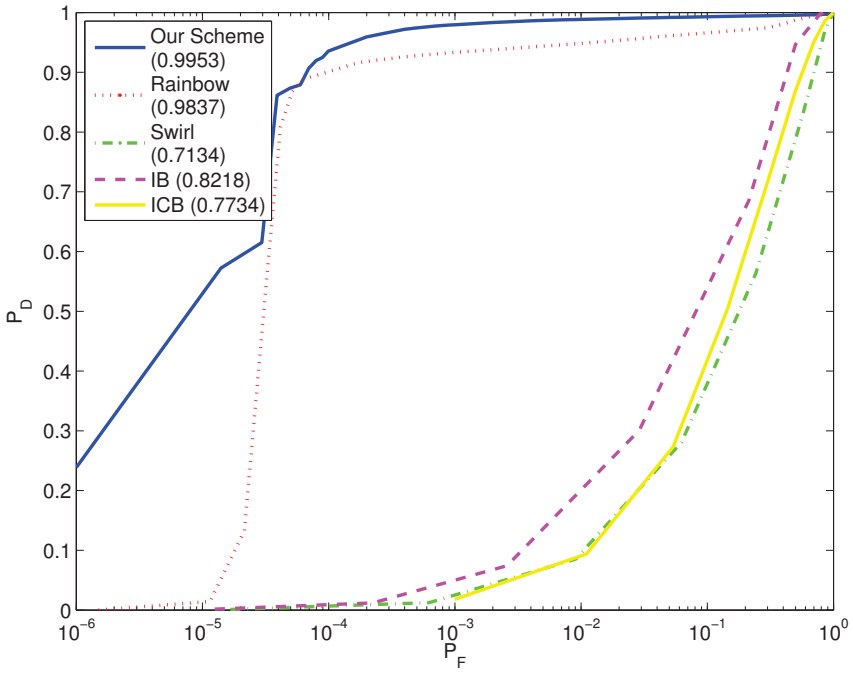


Figure 6.15: Comparison of algorithms on Scenario A with $n = 6$.

i.e., 500% chaff traffic added, $S = 2$ and random delays with $A_{max} = 50$ ms. As before, we fix $n = 51$ in Scenario A and $n = 251$ in Scenario B. Results are shown in Figures 6.17 and 6.18. Note that RAINBOW or SWIRL are not designed to be robust against an active attacker.

Our algorithm is more robust to the considered traffic modifications than the rest of schemes, for example, in Scenario B, we achieve $AUC = 0.9828$, while IBW achieves $AUC = 0.8842$, ICBW $AUC = 0.8350$, and for both RAINBOW and SWIRL $AUC < 0.6$. Recall also that we do not modify the flow, while the rest do.

Our scheme performs better than RAINBOW, which is also a non-blind detection, although it does not modify the IPDs. The improvement in performance is due to using a likelihood test (optimal) instead of a normalized correlation. Recall also that the IPDs have been restricted to be larger than 10 ms. Lifting this restriction would

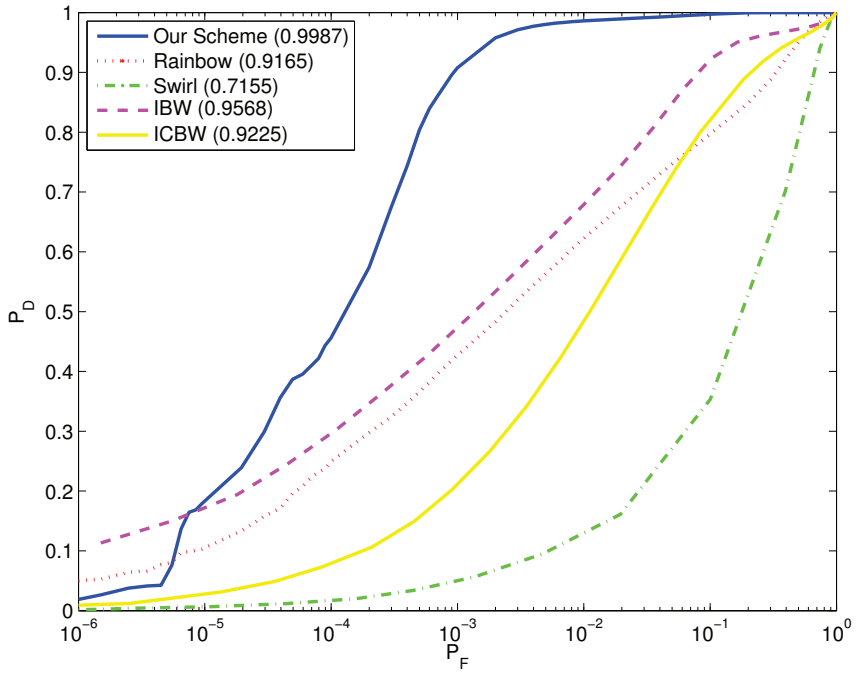


Figure 6.16: Comparison of algorithms on Scenario B with $n = 51$.

have a bigger impact on passive analysis than on a watermarking scheme.

6.8 Real Implementation

Obviously, simulations are not fully realistic. To check if simulator results are applicable to real networks, we carry out a real implementation of the proposed passive analysis scheme, the watermark modification proposed in Section 6.6 and the watermark schemes with which we compared in Section 6.7 for scenarios A and B.

For the first experiment, we launched three EC2 [122] instances. We used replayed SSH connections from real traces taken at University of Vigo and the stepping stone was created by forwarding the traffic with the `socat` command. For the second experiment, we replay connections from real HTTP traces also from University of Vigo. We use $n = 6$ packets and $n = 51$ for Scenarios A and B, respectively. The

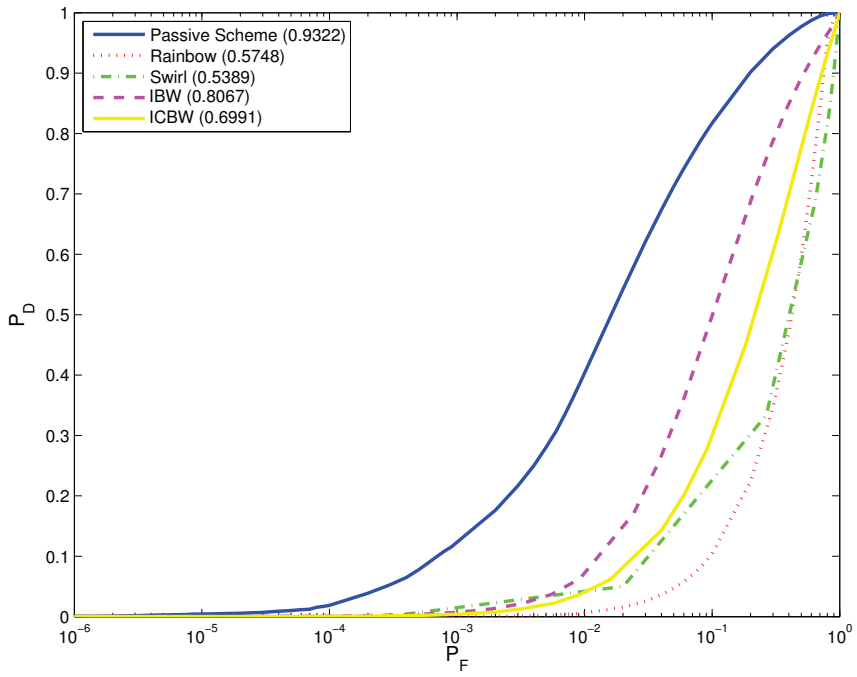


Figure 6.17: Comparison of algorithms on Scenario A under flow modification with $n = 51$.

experiment is repeated 1000 times in each case. In order to obtain values of the test under H_0, we use the saved timing information from the previous sequence in the non-blind cases, i.e., our proposed method and RAINBOW, while for the blind cases, i.e., IBW, ICBW and SWIRL, we use a different random key.

The chosen parameters are the maximum IPD variation for RAINBOW and a watermark modification of 5 ms in Scenario A and 20 ms in Scenario B, that are the median and maximum amplitudes in the experiments presented in [88]. For the blind-watermark, IBW and ICBW use an interval size of 500 ms, and SWIRL uses an interval length of 250 ms and 1000 ms for Scenarios A and B, respectively, divided into 5 subintervals of 3 slots each. These values have been chosen to maximize the AUC in each scenario.

Experiments are carried out in a non-active-attack scenario, this means that

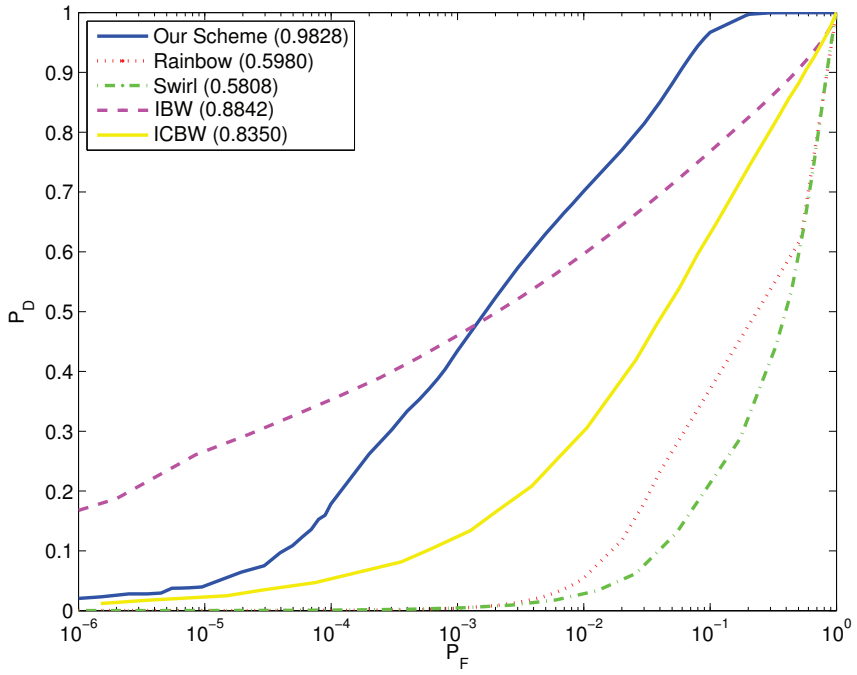


Figure 6.18: Comparison of algorithms on Scenario B under flow modification with $n = 251$.

insertions and losses are only due to repacketization. As the detector from(6.10) needs a value for P_L, we use P_{NL} from Table 6.1.

Results in Scenario A are similar to the simulator results: AUC=0.9987 for Real Scenario vs AUC=0.9983 for the Simulator. However, Scenario B shows a decrease in performance for the Real Scenario compared to the simulator results. This loss of performance affects all schemes, being for ours less severe.

6.9 Conclusions

In this chapter we have presented a highly-optimized traffic analysis method for deciding if two flows are linked which can be used as passive analysis. We also present a watermarking scheme based on this scheme. We develop an optimal decoder, i.e.,

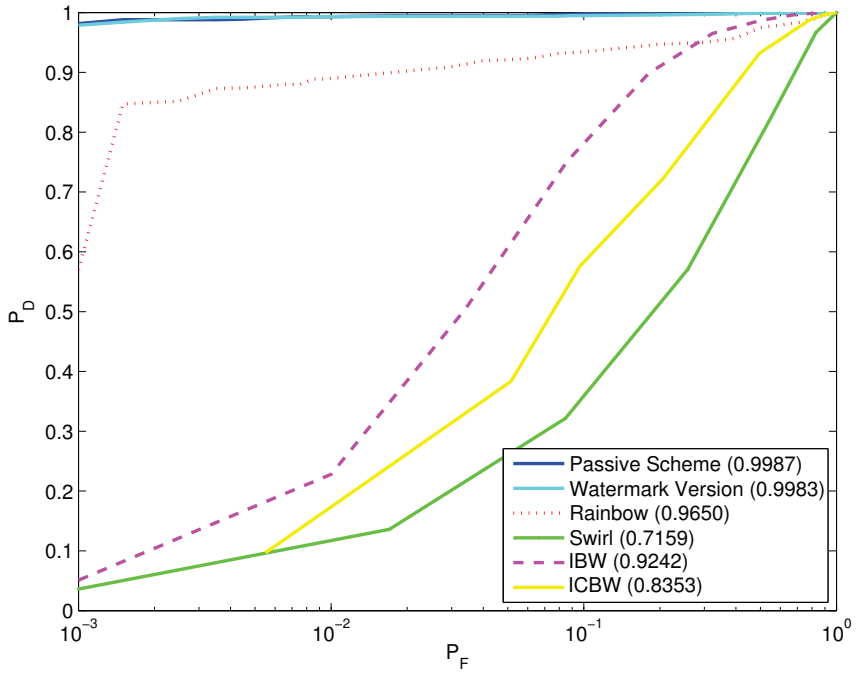


Figure 6.19: Real Implementation on Scenario A with $n = 6$.

the likelihood-ratio test, that allows to achieve a very good performance under a passive analysis scheme. For example, with 21 packets separated at least 10 ms we can correlate two flows, obtaining $P_D = 0.9861$, given a false alarm probability equal to 10^{-5}, without flow modifications.

A more robust detector is created that can deal with chaff traffic, flow splitting and random delays added by an attacker. To this end, packet matching is carried out by removing the packets that do not have a correspondent in the other flow. Then, a new likelihood-ratio test that considers losses and the maximum delay that an attacker can add is derived.

Afterwards, we study the trade-off between improvement and detectability of a watermarking scheme based on our algorithm. We also show a comparison with four state-of-the-art traffic watermarking schemes. Finally, a real implementation is carried out to show that the simulator results can be extended to real networks.

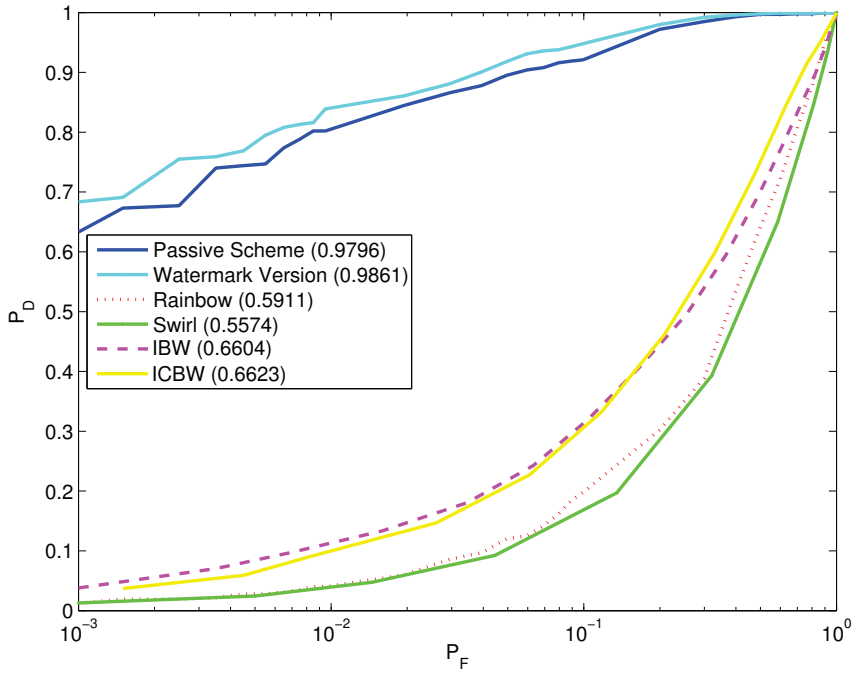


Figure 6.20: Real Implementation on Scenario B with $n = 51$.

The obtained results show that when dealing with independent flows, passive analysis schemes with an optimal detector can compete with and even outperform state-of-the-art traffic watermarking schemes, giving the advantages of being undetectable, which decreases the risk of a traffic modification attack, and that they can be carried out ex-post, in addition to in real-time. This makes it possible to use our schemes in forensic analysis applications as well as in intrusion detection.

Chapter 7

Flow-Correlation with an adversary

7.1 Introduction

We have seen in Chapter 6 how to correlate flows based on the IPDs, and we discussed the possibility that an adversary, such as a stepping stone or an anonymous network relay, may modify the flow to prevent the correlation by introducing delays or adding dummy packets to such flow. There, we dealt with an adversary that chooses its attack randomly instead of selecting the optimal attack. In this chapter, we deal with an adversary that has the largest impact on the correlation.

The aim of this chapter is to study the limits of passive traffic-analysis and flow fingerprinting in an adversarial environment. To achieve this goal, we propose a game-theoretic framework and look for the optimum strategies that the players should adopt. A similar game-theoretic framework has been used in other contexts such as Information Hiding [124], and Source Identification [125].

This game involves two players: the traffic analyst (TA) that tries to correlate

flows and the adversary (AD) whose goal is to impair this correlation.

We propose two different games depending on the characteristics of the flows we deal with: independent flows and correlated flows. In the first case, the TA wants to differentiate if the received flow is linked with a known flow or is a different flow without any relation with the known flow but coming from the same IPD distribution. For instance, a real scenario for this game is to differentiate between different SSH sessions. In the second game, the TA wants to decide between a fingerprinted flow and the same flow with no fingerprinted, an application can be to differentiate between clients that access the same web page.

7.2 Player order and equilibria

Recall from Section 2.2 that in sequential games, where players choose their actions in a given order, the subgame perfect equilibrium (SPE) is a refinement of the Nash Equilibrium that eliminates non-credible threats.

The SPE solution varies with the order in which players choose their actions, as a given player at the SPE knows which actions have taken place before his own (otherwise the player could improve his/her utility given this information). Hence, the solution to the game depends on the order in which the players choose their actions.

As the players choose their actions in a given order, an intuitive approach for a player (A) is to assume that the other player (B) will find out their action pattern and B decides his next action according to it, for instance this situation seems realistic when the detection is done off-line. Hence, the equilibrium of the game is:

$$u^* = \max_{w^n} \min_{A_{AD}} \max_{\Lambda_1} u(A_{TA}, A_{AD}), \tag{7.1}$$

that we represent with the superscript *. In the graphs the solution of this game is

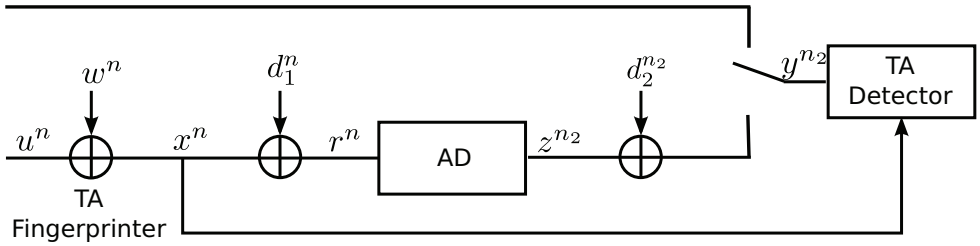


Figure 7.1: Model of the Flow Fingerprinting Game: independent flows

plotted with a solid line.

A more conservative approach for the TA is to assume that he is unable to know A_{AD} but the adversary is able to know TA's action pattern. Under this situation, the equilibrium is:

$$\underline{u} = \max_{w^n,\Lambda_1} \min_{A_{AD}} u(A_{TA}, A_{AD}), \tag{7.2}$$

which is represented with an underline. This scenario would model a real-time detection of stepping stones where the adversary knows the intrusion detection techniques used by the TA.

The most optimistic scenario for the TA is when the adversary behavior is known, in this case the equilibrium is:

$$\overline{u} = \min_{A_{AD}} \max_{w^n,\Lambda_1} u(A_{TA}, A_{AD}), \tag{7.3}$$

which is denoted using an overline. For instance, a real situation can be when the adversary is a Tor relay whose behavior is public.

As $\underline{u} \leq u^* \leq \overline{u}$, equality holds only when a saddle-point strategy exists. In a zero-sum a saddle-point is always a Nash equilibrium (NE) [126].

7.3 Flow fingerprinting game: independent flows

The flow fingerprinting game for independent flows is represented in Figure 7.1. In this game, the TA has to correctly decide whether the candidate flow (y^{n_2}) is the known flow (u^n) or a different one. This is the problem dealt with in Chapter 6. Formally, the task of the TA is to accept or reject the hypothesis that y^{n_2} is indeed the same flow as u^n.

To improve the efficiency, the TA can modify the flow by embedding a fingerprint w^n. Due to the nature of the problem, the modification must be additive, i.e., $x^n = u^n + w^n$. We constrain the fingerprint to delay any packet at most W_{max} seconds. This flow will suffer a network delay of d_1^n before reaching the adversary. We denote by r^n the flow received by the adversary.

The goal of the adversary is to modify the flow, producing z^{n_2}, in such a way that the detector decides that this sequence is not related with u^n. In order to do this, the adversary can modify the flow by adding delays and dummy packets. We denote by z^{n_2} the output flow of the adversary. This flow suffers an additional delay $d_2^{n_2}$ due to the network between the adversary and the detector.

We represent by D the delay suffered by a packet in the whole path, i.e., $D = D_1 + D_2$. Recall that ΔD is the PDV or jitter.

We assume that $f_{\Delta U}(\Delta u)$ is known by both players and define the hypotheses:

$$H_0 : y^{n_2} \text{ and } u^n \text{are not linked}$$
$$H_1 : y^{n_2} \text{ and } u^n \text{are linked.}$$

We define the flow fingerprinting game for independent flows as follows:

Definition 1. The $FFG_I(A_{TA}; A_{AD}; u)$ is a zero-sum game played by the TA and the adversary, where

- The set of actions the TA can choose from, i.e. A_{TA}, is the duple of possible fingerprint sequences w^n and acceptance regions Λ_1:

$$A_{TA} = \{w^n \times \Lambda_1 : 0 \leq w_i \leq W_{max},\ i \in [1, n]\} \tag{7.4}$$

- The set of possible attacks that the adversary can choose from $A_{AD} = \{f(z^{n_2}|r^n)\}$. As we will study two different adversaries, we will formally define this function in each section.

- We use two different utility functions. The first utility function, u_D is the probability of detection, P_D, for which the probability of false positive, P_F, is below a certain threshold η:

$$u_D(A_{TA}, A_{AD}) = Pr(Y^{n_2} \in \Lambda_1|H_1) \wedge Pr(Y^{n_2} \in \Lambda_1|H_0) < \eta. \tag{7.5}$$

  The second utility function we use, denoted by u_A, is the AUC:

$$u_A(A_{TA}, A_{AD}) = \int_0^1 Pr(Y^{n_2} \in \Lambda_1|H_1)d\eta, \text{ where } \eta = Pr(Y^{n_2} \in \Lambda_1|H_0). \tag{7.6}$$

7.3.1 Optimal Detector

As for directly correlating timing sequences a precise estimate of f_D is required, and this is in some cases difficult to obtain, the use of the difference timing sequence, that is, IPDs, seems more reasonable.

The optimal detector, according to Neyman-Pearson Lemma, is the likelihood ratio test:

$$\begin{aligned}\Lambda_1(y^{n_2}, x^n, \hat{f}(z^{n_2}|r^n)) = \int_{\mathcal{R}^{+n}} \int_{\mathcal{R}^{+n_2}} & \frac{f_{\Delta D_2^{n_2}}(\Delta(y^{n_2} - z^{n_2}))\hat{f}(z^{n_2}|r^n)}{f_{\Delta U^{n_2}}(\Delta y^{n_2})} \\ & \cdot f_{\Delta D_1^n}(\Delta(r^n - x^n))dz^{n_2}dr^n\end{aligned} \tag{7.7}$$

where $\hat{f}(z^{n_2}|r^n)$ is the assumed distribution of $f(z^{n_2}|r^n)$ by the detector. If we are under condition 7.1 and 7.3, then $\hat{f}(z^{n_2}|r^n) = f(z^{n_2}|r^n)$.

7.4 Delaying adversary

In this section we derive the detector for the case when the adversary is limited to delaying the packets up to A_{max} seconds, without adding or removing any packets from the flow. Under this condition, there exists a one-to-one correspondence between x^n and y^{n_2}, i.e., $n_2 = n$. Formally, the adversary's actions can be defined as:

$$A_{AD} = \{f(a^n|r^n) : 0 \leq a_i \leq A_{max},\ i \in [1, n]\}, \tag{7.8}$$

where $a^n = z^n - r^n$ is the sequence of delays added by the adversary.

From (7.7), the likelihood ratio test under this adversary becomes:

$$\Lambda_1(y^n, x^n, \hat{f}_{A^n|X^n}) = \int_{\mathcal{R}^n} \frac{f_{\Delta D^n}(\Delta(y^n - x^n - a^n))\hat{f}_{A^n|X^n}(a^n|x^n)}{f_{\Delta U^n}(\Delta y^n)} da^n. \tag{7.9}$$

Restricting the detector (7.9) to first order statistics, we obtain

$$\Lambda_1(y^n, x^n, \hat{f}_{A^n|X^n}) = \prod_{i=1}^{n-1} \frac{\int_{\mathcal{R}} f_{\Delta D}(\Delta d_i + z))\hat{f}_{A_{i+1}-A_i|X^n}(z|x^n)dz}{f_{\Delta U}(\Delta y_i)} \tag{7.10}$$

where $\hat{f}_{A_{i+1}-A_i|X^n}(x) = \int_0^{A_{max}} \hat{f}_{A_{i+1}}(y)\hat{f}_{A_i}(y - x)dy$.

The solutions of the different games are:

$$u_D^* = \max_{w^n} \min_{f_{A^n|x^n}} Pr(\Lambda_1(x^n + A^n + D^n, x^n, f_{A^n|x^n}) > \epsilon) \tag{7.11}$$

$$\underline{u}_D = \max_{w^n, \hat{f}_{A^n|x^n}} \min_{f_{A^n|x^n}} Pr(\Lambda_1(x^n + A^n + D^n, x^n, \hat{f}_{A^n|x^n}) > \epsilon) \tag{7.12}$$

$$\overline{u}_D = \min_{f_{A^n|u^n}} \max_{w^n} Pr(\Lambda_1(x^n + A^n + D^n, x^n, f_{A^n|u^n}) > \epsilon) \tag{7.13}$$

Unfortunately, calculating the solutions given by (7.11), (7.12) or (7.13) are still intractable problems if we allow any kind of delay distribution.

7.4.1 Deterministic Attack

The simplest adversary we can imagine is one that acts in a deterministic way. This means that if he receives two identical sequences the output will be the same, i.e $a^n = f(r^n)$. Hence, $f_{A^n|\Delta r^n}(x^n) = \prod_{i=1}^{n} \delta(x_i - a_i)$ and $\hat{f}_{A^n|\Delta r^n}(x^n) = \prod_{i=1}^{n} \delta(x_i - \hat{a}_i)$.

The solutions of this game are:

$$u^* = \max_{w^n} \min_{a^n} Pr\left(\prod_{i=1}^{n-1} \frac{f_{\Delta D}(\Delta_i d^n)}{f_{\Delta U}(\Delta_i(u^n + w^n + a^n + d^n))} > \epsilon \right) \tag{7.14}$$

$$\underline{u} = \max_{w^n, \hat{a}^n} \min_{a^n} Pr\left(\prod_{i=1}^{n-1} \frac{f_{\Delta D}(\Delta_i(d^n + a^n - \hat{a}^n))}{f_{\Delta U}(\Delta_i(u^n + w^n + a^n + d^n))} > \epsilon \right) \tag{7.15}$$

$$\overline{u} = \min_{a^n} \max_{w^n} Pr\left(\prod_{i=1}^{n-1} \frac{f_{\Delta D}(\Delta_i d^n)}{f_{\Delta U}(\Delta_i(u^n + w^n + a^n + d^n))} > \epsilon \right) \tag{7.16}$$

Note that when $\hat{a}^n = a^n$ (i.e., when the detector is chosen after the adversary) the adversary delays are the ones that maximize the likelihood of $x^n + a^n + d^n$ coming from the distribution of U^n, i.e., making the sequence as typical as possible. Conversely, the fingerprint delays are the ones that minimize this likelihood.

Performance

We calculate the solution of the game for the same two scenarios studied in Section 6.4.1 using $n = 5$ and $n = 10$ for scenarios 1 and 2, respectively.

We evaluate 4 different conditions:

- A_1: Passive analysis without attack ($A_{max} = W_{max} = 0$ ms)
- A_2: Passive analysis with attack ($A_{max} = 150$ ms and $W_{max} = 0$ ms).
- A_3: Passive analysis with attack ($A_{max} = 250$ ms and $W_{max} = 0$ ms).
- A_4: Fingerprint with attack ($A_{max} = 250$ ms and $W_{max} = 100$ ms).

Note that $A_{max} - W_{max}$ is identical in A_2 and A_4. Also we do not plot $\overline{u}$ in A_2 and A_3 as it is identical to $u*$ due to the lack of fingerprint.

Results are depicted in Figures 7.2 and 7.3 from which we can draw several conclusions. First, if we focus on the case where both fingerprint and attack exist, i.e., A_4 conditions, we see a different behavior for u^* between both scenarios; in Scenario 1, the ROC for u^* is nearer to $\overline{u}$ than to $\underline{u}$, while in Scenario 2 the performance for u^* is very similar to $\underline{u}$. The reason for this different behavior is that in Scenario 1 the variation of the IPDs comes basically from the attack delay and not from the network variability, then the detector knowing the attack becomes the predominant factor, while in Scenario 2 the PDV is comparable to adversary's delays so the knowledge of the attack is not so important for the detector.

Second, comparing A_2 and A_4 we see that in Scenario 1 the A_4 situation is beneficial for the TA if the detector knows the attack, i.e., u^* and $\overline{u}$, but is beneficial for the adversary if the attack is chosen knowing the detector actions, i.e., $\underline{u}$. In Scenario 2, the results of A_2 and A_4 are very similar unless the fingerprint is selected knowing the attack function, i.e., $\overline{u}$.

7.4.2 Truncated-Gaussian Attack

The previous model gives the chance to the other player to compensate it in the next action. For this reason, players may be interested in randomizing their actions, i.e., given the same input, the output can vary.

We study the case when the distribution of delays follows a truncated Gaussian in the allowed interval. Hence, $a_i \sim N(\mu_{a,i}, \sigma_a^2 | 0 \leq a_i \leq A_{max})$ and $w_i \sim N(\mu_{w,i}, \sigma_w^2 | 0 \leq a_i \leq W_{max})$. Note that this model includes as extreme cases the deterministic attack seen previously (i.e., $\sigma^2 \to 0$), and a uniform attack (i.e., $\sigma^2 \to \infty$). The detector comes from (7.10) where the attack delays are a truncated Gaussian with guessed

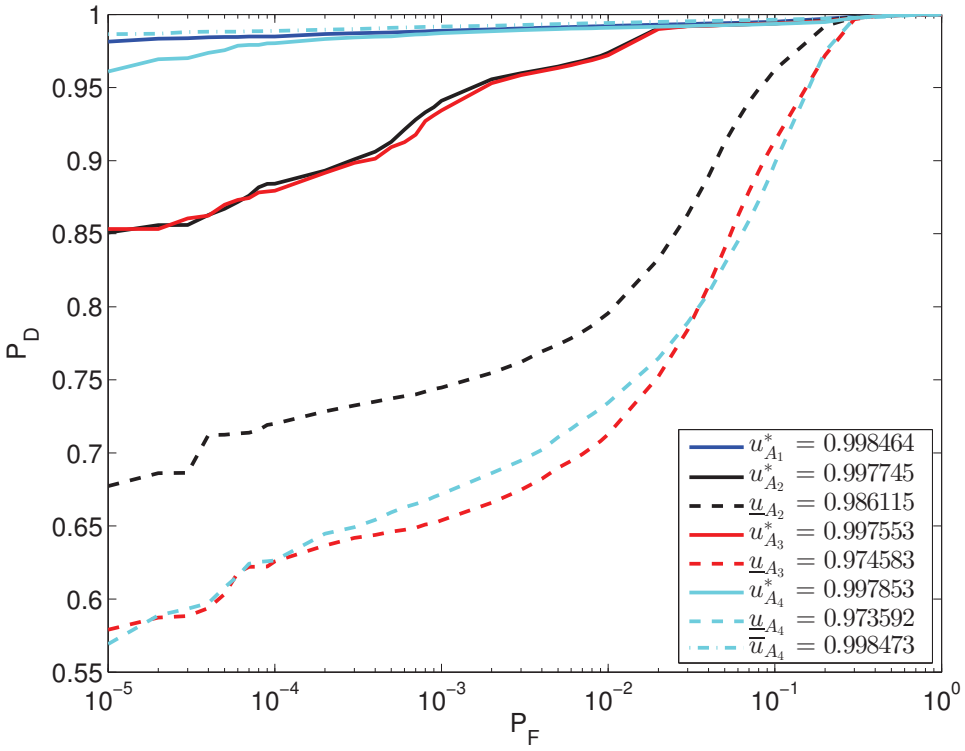


Figure 7.2: Solution of the constant games for Scenario 1 ($n = 5$).

parameters $\hat{\mu}_a^n$ and $\hat{\sigma}_a^2$.

Formally, the actions available to each player are:

$$A_{TA} = \{\mu_w^n \times \sigma_w^2 \times \hat{\mu}_a^n \times \hat{\sigma}_a^2\} \tag{7.17}$$

$$A_{AD} = \{\mu_a^n \times \sigma_a^2\} \tag{7.18}$$

Since the simultaneous optimization with respect to all the variables is computationally unfeasible, we compute first the solution for the mean sequence assuming zero variance, i.e. the solution of the constant game shown in equations (7.14), (7.15) and (7.16) for u^*, $\underline{u}$ and $\overline{u}$, respecitvely. Afterwards, we calculate the variance at the SPE without modifying the mean sequence. We solve the two scenarios presented previously for each of the proposed games. Tables 7.1, 7.2, 7.3, 7.4, 7.5, and 7.6 show the u_A depending on the parameters and we represent the solution of the game in

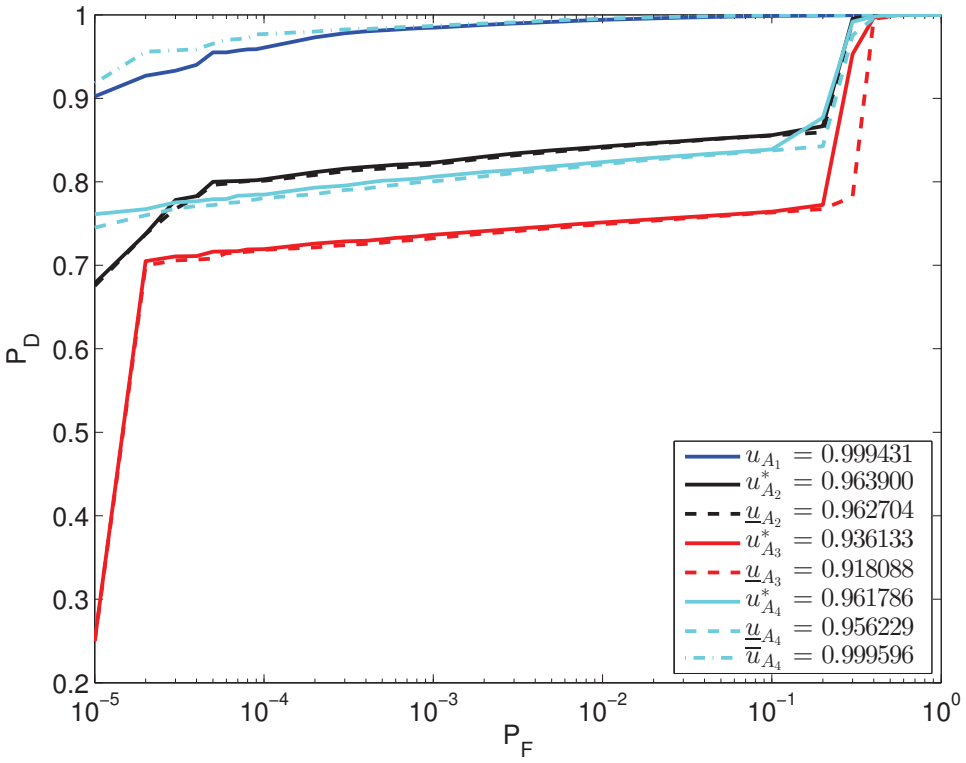


Figure 7.3: Solution of the constant games for Scenario 2 ($n = 10$).

boldface.

The fingerprint variance, σ_f^2, is very small in most of the cases and when it is a bit larger the utility of choosing the smallest variance is reduced only by a small quantity. This means that a deterministic fingerprint is optimal or quasi-optimal.

In the case that the fingerprint is chosen without knowing the attack distribution parameters, i.e., u^* and $\underline{u}$ conditions, the adversary should choose a variance in the interval $[10^{-3}, 10^{-2}] \cdot A_{max}$. This small variance makes the attack virtually deterministic (i.e., $\sigma \to 0$), implying that making the output sequence more typical prevails over increasing the uncertainty of a^n for the detector. When the TA fingerprinter knows the behavior of the adversary, i.e., $\overline{u}$, then the latter is forced to increase his variance to 10^{-1} to prevent the TA from choosing the fingerprint that is less affected

Table 7.1: $u_A^*(\sigma_f, \sigma_a)$ for Scenario 1 ($n = 5$, $A_{max} = 250$ms and $W_{max} = 100$ms).

σ_a/A_{max} \ σ_f/W_{max}	10^{-4}	10^{-3}	10^{-2}	10^{-1}	1
10^{-4}	0.9978	0.9987	0.9963	0.9960	0.9968
10^{-3}	0.9900	0.9891	0.9878	0.9873	0.9727
10^{-2}	0.9418	**0.9447**	0.9428	0.9415	0.9319
10^{-1}	0.9710	0.9733	0.9686	0.9721	0.9774
1	0.9977	0.9982	0.9972	0.9967	0.9981
10	0.9979	0.9985	0.9965	0.9977	0.9982

Table 7.2: $u_A^*(\sigma_f, \sigma_a)$ for Scenario 2 ($n = 10$, $A_{max} = 250$ms and $W_{max} = 100$ms).

σ_a/A_{max} \ σ_f/W_{max}	10^{-4}	10^{-3}	10^{-2}	10^{-1}	1
10^{-4}	0.9333	0.9396	0.9360	0.9252	0.9043
10^{-3}	0.9131	0.9223	0.9181	0.9051	0.8843
10^{-2}	0.9086	**0.9129**	0.9094	0.9001	0.8809
10^{-1}	0.9487	0.9496	0.9454	0.9458	0.9465
1	0.9988	0.9991	0.9991	0.9988	0.9988
10	0.9990	0.9991	0.9990	0.9991	0.9989

by the attack.

We also see that when the detector does not know the parameter of the adversary, i.e. $\underline{u}$, the detector has to overestimate the variance chosen by the adversary, for instance choosing $\hat{\sigma}_a = 10^{-1}$ instead of 10^{-2}, so that the adversary is not able to impair the correlation by selecting a high variance attack.

Table 7.3: $\underline{u}_A(\sigma_a, \hat{\sigma}_a)$ for Scenario 1 ($n = 5$, $A_{max} = 250$ms and $W_{max} = 100$ ms).

$\hat{\sigma}/A_{max}$ \ σ/A_{max}	10^{-4}	10^{-3}	10^{-2}	10^{-1}	1	10
10^{-4}	0.9708	0.9389	0.8955	0.8936	0.8475	0.8448
10^{-3}	0.9751	0.9481	0.9010	0.8958	0.8514	0.8532
10^{-2}	0.9754	0.9581	0.9043	0.9047	0.8836	0.8816
10^{-1}	0.9581	0.9532	**0.9369**	0.9641	0.9813	0.9773
1	0.9357	0.9414	0.9579	0.9949	0.9991	0.9987
10	0.9345	0.9429	0.9618	0.9954	0.9989	0.9987

Table 7.4: $\underline{u}_A(\sigma_a, \hat{\sigma}_a)$ for Scenario 2 ($n = 10$, $A_{max} = 250$ms and $W_{max} = 100$ms).

$\hat{\sigma}_a/A_{max}$ \ σ_a/A_{max}	10^{-4}	10^{-3}	10^{-2}	10^{-1}	1	10
10^{-4}	0.9236	0.9087	0.9061	0.9122	0.9435	0.9494
10^{-3}	0.9231	0.9096	0.9085	0.9109	0.9421	0.9450
10^{-2}	0.9215	0.9100	0.9076	0.9139	0.9473	0.9484
10^{-1}	0.9138	**0.9082**	0.9118	0.9485	0.9856	0.9895
1	0.9032	0.9043	0.9277	0.9893	0.9989	0.9988
10	0.9027	0.9039	0.9259	0.9882	0.9988	0.9986

Table 7.5: $\overline{u}_A(\sigma_f, \sigma_a)$ for Scenario 1 ($n = 5$, $A_{max} = 250$ms and $W_{max} = 100$ms).

σ_a/A_{max} \ σ_f/W_{max}	10^{-4}	10^{-3}	10^{-2}	10^{-1}	1
10^{-4}	0.9982	0.9990	0.9969	0.9968	0.9987
10^{-3}	0.9983	0.9991	0.9970	0.9967	0.9988
10^{-2}	0.9982	0.9989	0.9968	0.9965	0.9977
10^{-1}	0.9960	**0.9975**	0.9950	0.9946	0.9931
1	0.9982	0.9985	0.9964	0.9968	0.9983
10	0.9980	0.9982	0.9962	0.9967	0.9983

Table 7.6: $\overline{u}_A(\sigma_f, \sigma_a)$ for Scenario 2 ($n = 10$, $A_{max} = 250$ms and $W_{max} = 100$ms).

σ_a/A_{max} \ σ_f/W_{max}	10^{-4}	10^{-3}	10^{-2}	10^{-1}	1
10^{-4}	0.9989	0.9991	0.9994	0.9985	0.9990
10^{-3}	0.9989	0.9991	0.9993	0.9983	0.9984
10^{-2}	0.9984	0.9984	0.9986	0.9971	0.9963
10^{-1}	0.9934	**0.9949**	0.9943	0.9922	0.9915
1	0.9989	0.9991	0.9991	0.9984	0.9989
10	0.9991	0.9993	0.9991	0.9988	0.9990

7.4.3 Other distribution attacks

In this section, we evaluate whether the game results change significantly when other different distributions are used. Concretely, we evaluate the truncated Laplace and the truncated Cauchy distributions. We solve the game in the same way as presented in the previous section. We depict the ROC at the SPE for the proposed distribution in Figures 7.4 and 7.5, and the scale parameter at the SPE in Tables 7.7 and 7.8.

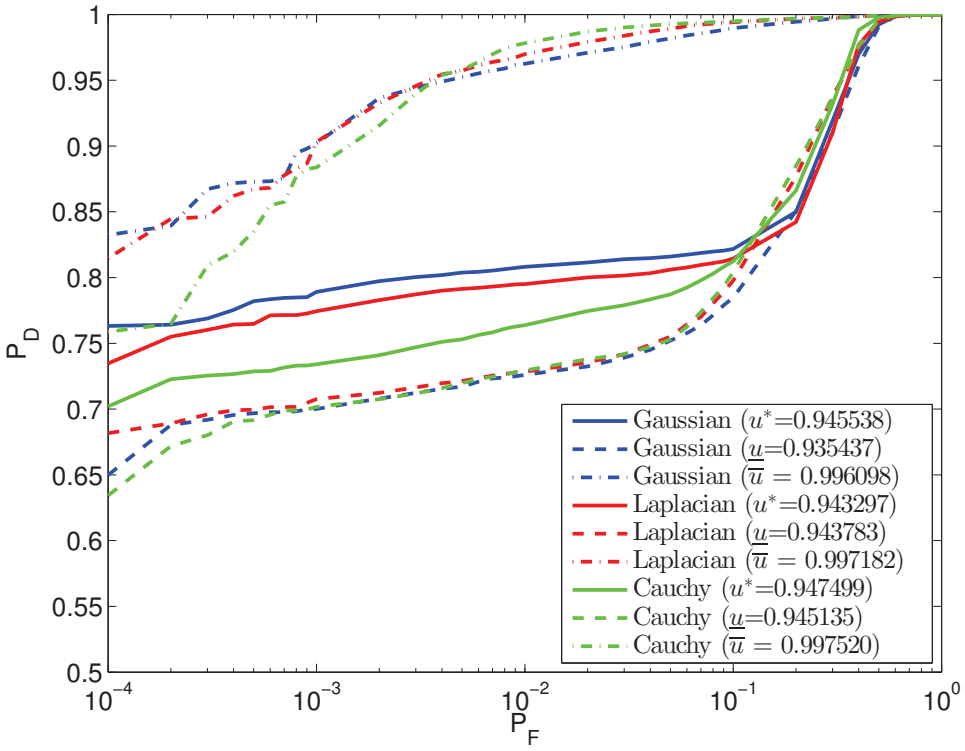


Figure 7.4: Solution of the truncated Gaussian (blue), truncated Laplace (red) and truncated Cauchy (green) games for Scenario 1 ($n = 5$, $A_{max} = 250$ms and $W_{max} = 100$ms).

We see that the difference among the distributions is not significant, but in any case the Gaussian distribution is the one that impairs slightly more the correlation based on the AUC.

7.4.4 Distribution mismatch between the adversary and the decoder

In this section, we evaluate the consequences of an adversary who not only chooses the distribution parameters but also the distribution itself. Obviously, only differs from the previous one under $\underline{u}$, in which the adversary uses an attack distribution different than that the assumed at the detector.

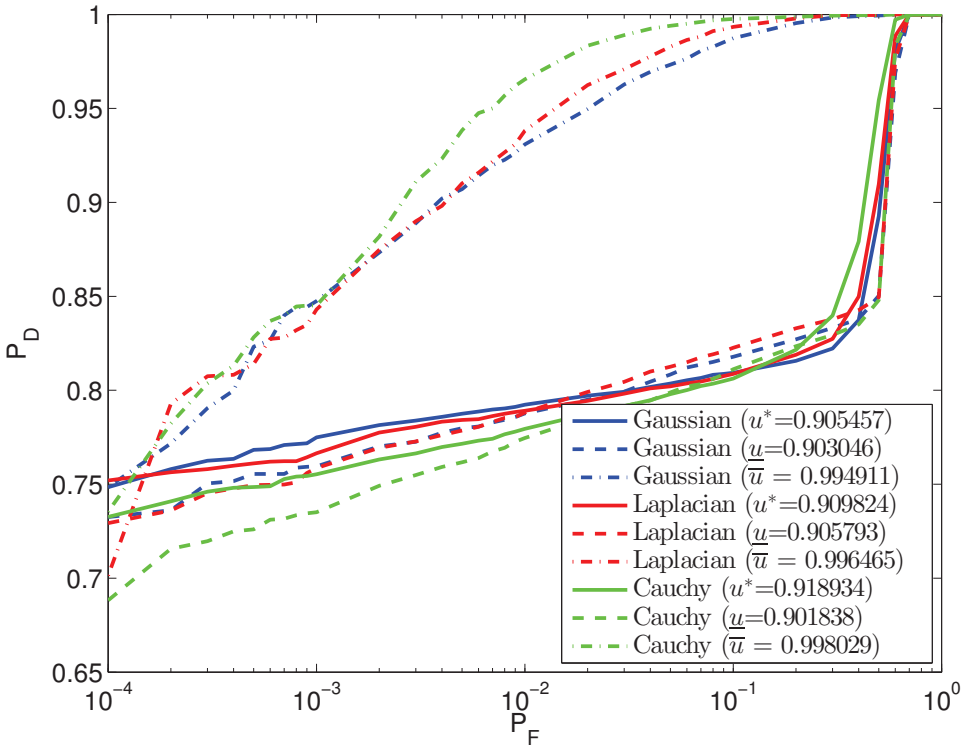


Figure 7.5: Solution of the truncated Gaussian (blue), truncated Laplace (red) and truncated Cauchy (green) games for Scenario 2 ($n = 20$, $A_{max} = 250$ms and $W_{max} = 100$ms).

Table 7.7: Scale parameter at the SPE for different distribution attacks for Scenario 1 ($n = 5$, $A_{max} = 250$ms and $W_{max} = 100$ms).

	u_A^*	$\underline{u}_A$	$\overline{u}_A$
Truncated Gaussian	$\sigma_F = 10^{-3}$ $\sigma_A = 10^{-2}$ $\hat{\sigma}_A = 10^{-2}$	$\sigma_F = 10^{-3}$ $\sigma_A = 10^{-2}$ $\hat{\sigma}_A = 10^{-1}$	$\sigma_F = 10^{-3}$ $\sigma_A = 10^{-1}$ $\hat{\sigma}_A = 10^{-1}$
Truncated Laplace	$\sigma_F = 10^{-3}$ $\sigma_A = 10^{-2}$ $\hat{\sigma}_A = 10^{-2}$	$\sigma_F = 10^{-3}$ $\sigma_A = 10^{-2}$ $\hat{\sigma}_A = 10^{-1}$	$\sigma_F = 10^{-4}$ $\sigma_A = 10^{-1}$ $\hat{\sigma}_A = 10^{-1}$
Truncated Cauchy	$\sigma_F = 10^{-3}$ $\sigma_A = 10^{-2}$ $\hat{\sigma}_A = 10^{-2}$	$\sigma_F = 10^{-3}$ $\sigma_A = 10^{-3}$ $\hat{\sigma}_A = 10^{-1}$	$\sigma_F = 10^{-3}$ $\sigma_A = 10^{-1}$ $\hat{\sigma}_A = 10^{-1}$

We assume that the detector is designed using a Gaussian-distributed attack as it is the distribution that impairs slightly more the detector, and we evaluate the

Table 7.8: Scale parameter at the SPE for different distribution attacks for Scenario 2 ($n = 10$, $A_{max} = 250$ms and $W_{max} = 100$ms).

	u_A^*	$\underline{u}_A$	$\overline{u}_A$
Truncated Gaussian	$\sigma_F = 10^{-3}$ $\sigma_A = 10^{-2}$ $\hat{\sigma}_A = 10^{-2}$	$\sigma_F = 10^{-3}$ $\sigma_A = 10^{-3}$ $\hat{\sigma}_A = 10^{-1}$	$\sigma_F = 10^{-3}$ $\sigma_A = 10^{-1}$ $\hat{\sigma}_A = 10^{-1}$
Truncated Laplace	$\sigma_F = 10^{-2}$ $\sigma_A = 10^{-2}$ $\hat{\sigma}_A = 10^{-2}$	$\sigma_F = 10^{-3}$ $\sigma_A = 10^{-3}$ $\hat{\sigma}_A = 10^{-1}$	$\sigma_F = 10^{-4}$ $\sigma_A = 10^{-1}$ $\hat{\sigma}_A = 10^{-1}$
Truncated Cauchy	$\sigma_F = 10^{-4}$ $\sigma_A = 10^{-3}$ $\hat{\sigma}_A = 10^{-3}$	$\sigma_F = 10^{-3}$ $\sigma_A = 10^{-3}$ $\hat{\sigma}_A = 10^{-1}$	$\sigma_F = 10^{-3}$ $\sigma_A = 10^{-2}$ $\hat{\sigma}_A = 10^{-2}$

performance of the other two attack distributions used in the previous section. We simulate the same scenarios as in previous sections, and we depict the results in Figures 7.6 and 7.7 for Scenario 1 and 2 respectively. We can see that even the TA is known to decode assuming a truncated Gaussian the adversary does not increase significantly the impairment to the correlation by choosing a different distribution attack, allowing us to conclude that the location and scale parameters, i.e., mean and variance, are the predominant factors in the detector rather than the assumed attack distribution.

7.5 Chaff traffic adversary

In this section we derive the detector when the adversary not only can delay packets as we have assumed so far but he can also add certain amount of chaff traffic, making $n_2 \geq n$. Formally, the adversary actions can be defined as:

$$\begin{aligned} A_{AD} =&\{f(z^{n_2}|r^n) : \exists c^{n_A} \mid 0 \leq a_i \leq A_{max} \ \wedge \ \frac{n_A}{n} \leq P_A \wedge \ 0 \leq c_j \leq r_n + A_{max} \\ &\wedge \ z^{n_2} = \text{sort}((r^n + a^n)||c^{n_A}), i \in [1, n] \text{ and } j \in [1, n_A]\}, \end{aligned} \tag{7.19}$$

where c^{n_A} is the sequence of chaff packets, P_A is the maximum ratio between chaff and real traffic, $||$ represents the concatenation of sequences, and $\text{sort}(x^n)$ is a function

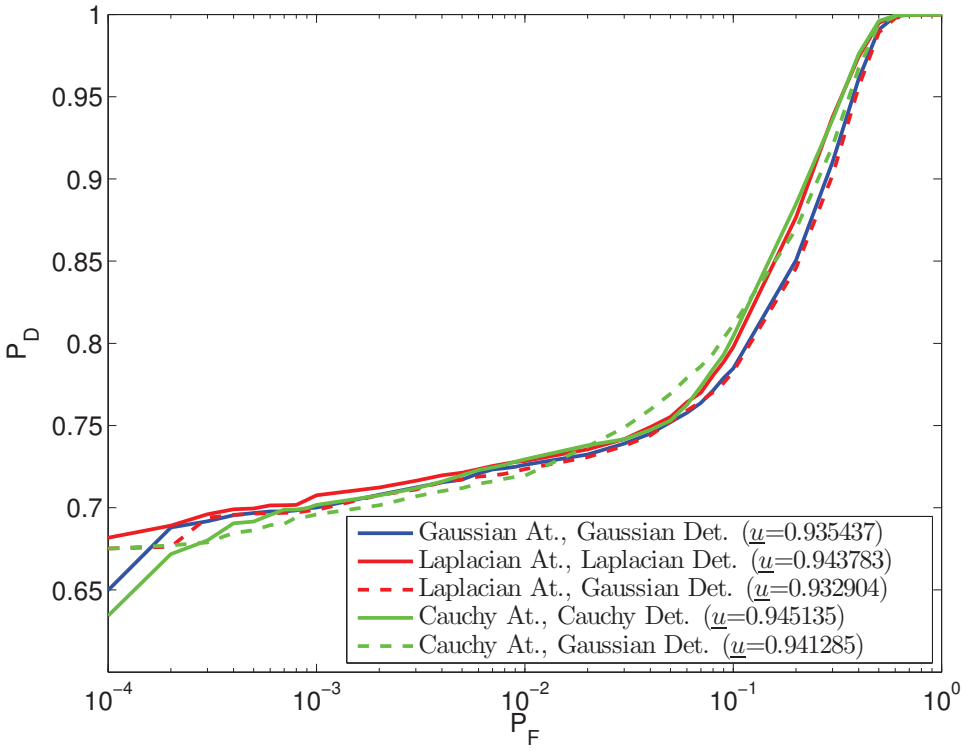


Figure 7.6: ROC curves using Gaussian detector for Scenario 1 ($n = 5$, $A_{max} = 250$ms and $W_{max} = 100$ms).

that returns a sorted version of the input sequence.

A Neyman-Pearson detector when $n_2 \neq n$ has to be based on the joint distribution as shown in (7.7) and this test becomes intractactable. For this reason, we implement the detector in two steps: first, a matching process takes place that outputs two sequences of the same size, and then we use the same likelihood test constructed in the previous section.

7.5.1 Matching Process

When dummy packets are added, i.e., $P_A > 0$, there does not exist a one-to-one relation between the flows x^n and y^{n_2}. To deal with this problem, we match each

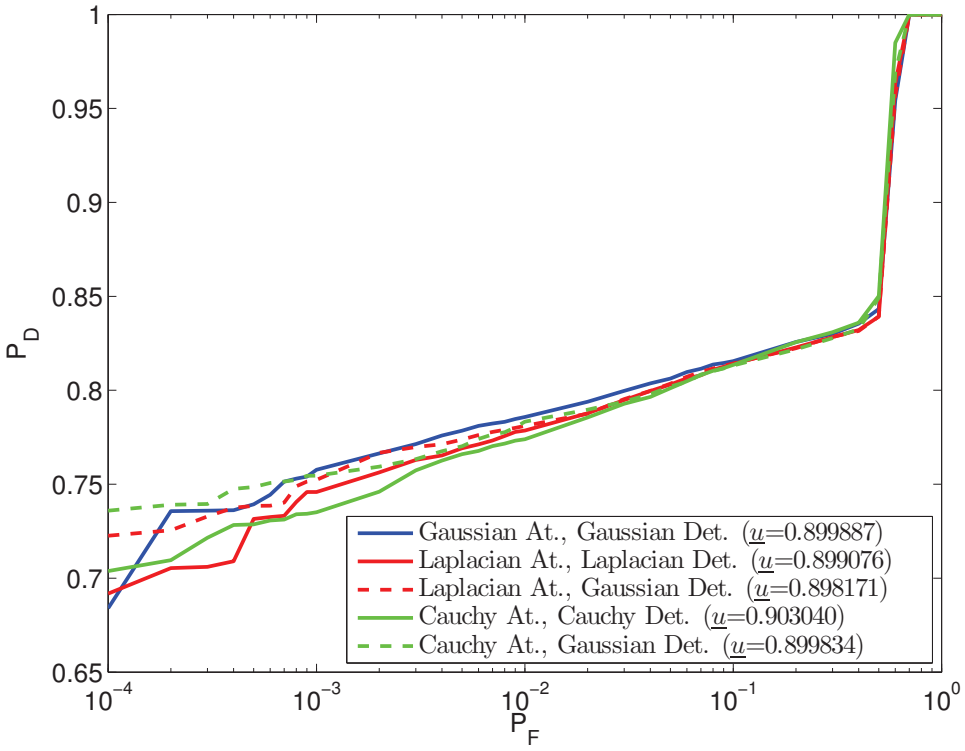


Figure 7.7: ROC curves using Gaussian detector for Scenario 2 ($n = 10$, $A_{max} =$ 250ms and $W_{max} = 100$ms).

packet of x^n with the most likely from y^{n_2}, later removing those packets of y^{n_2} that have no correspondence in x^n.

We represent the fact that the ith packet from x^n is paired with the jth packet from y^{n_2} by $m(i) = j$. Let $\mathcal{M}$ be the set of all injective functions from $\mathcal{N} = \{1, \ldots, n\}$ to $\mathcal{N}_2 = \{1, \ldots, n_2\}$, i.e., $\forall i_1, i_2 \in \mathcal{N},\ m(i_1) = m(i_2) \implies i_1 = i_2$. Then the matching function $m(x^n, y^{n_2})$ is the function from $\mathcal{M}$ that minimizes the mean square error between x^n and a shifted version of y^{n_2} as follows:

$$m = \arg\min_{\mathcal{M}} \sum_{i=1}^{n} (y_{p(i)} - x_i - \rho - E(a_i))^2, \tag{7.20}$$

where $E(a_i)$ is the expected value for the delay added by the adversary to the ith packet, and ρ is a synchronization constant equal to the sample mean of the delays,

i.e. $\rho = \frac{1}{n}\sum_{i=1}^{n} d_i$. In a real implementation, where the sample mean is unknown, ρ can be obtained through an exhaustive search (self-synchronization property) as shown in Chapter 6.

7.5.2 Chaff traffic of the adversary

We assume that the matching process selects those packets that give a higher value for the detector, i.e., $\Lambda_1(m(x^n, y^{n_2}), x^n, f_{A^n|\Delta X^n}) > \Lambda_1(m'(x^n, y^{n_2}), x^n, f_{A^n|\Delta X^n})$, $\forall m' \in \mathcal{M}$. Under this assumption, the adversary has to choose c^{n_A} so that these dummy packets are removed in the matching process. On the other hand, the adversary needs these packets to force the TA to consider longer possible sequences for y^{n+n_A}, as longer sequences of Y^{n_2} will increase ϵ for a given P_F.

7.5.3 Results

In this section, we obtain the solution of the proposed games when the adversary can also add chaff traffic, so we extend the simulator to handle it. We use the truncated Gaussian version of the game, as in the previous section we showed that among the attack distributions there is not a significant difference on the performance.

We show the utility, u_A, at the SPE for the proposed distribution in Tables 7.9 and 7.10, and the scale parameter at the SPE in Tables 7.11 and 7.12. As expected, the utility decreases with the amount of chaff, but the amount of traffic that would drop the correlation is very large, for instance in Scenario 2 u_A^* only drops from 0.9810 to 0.9649 using 1000% of chaff traffic. We also notice that under larger amount of chaff traffic the variance of the attack for $\overline{u}$ increases towards a uniform attack.

Table 7.9: Comparison of the utility at the SPE for different amount of chaff traffic for Scenario 1 ($n = 5$, $A_{max} = 250$ms and $W_{max} = 100$ms).

	u_A^*	$\underline{u}_A$	$\overline{u}_A$
No Chaff	0.9732	0.9669	0.9990
$P_A = 1$	0.9650	0.9555	0.9977
$P_A = 10$	0.9328	0.9224	0.9880

Table 7.10: Comparison of the utility at the SPE for different amount of chaff traffic for Scenario 2 ($n = 10$, $A_{max} = 250$ms and $W_{max} = 100$ms).

	u_A^*	$\underline{u}_A$	$\overline{u}_A$
No Chaff	0.9810	0.9780	0.9988
$P_A = 1$	0.9764	0.9739	0.9963
$P_A = 10$	0.9649	0.9655	0.9926

Table 7.11: Scale parameter at the SPE for different amount of chaff traffic for Scenario 1 ($n = 10$, $A_{max} = 250$ms and $W_{max} = 100$ms).

	u_A^*	$\underline{u}_A$	$\overline{u}_A$
No chaff	$\sigma_F = 10^{-4}$	$\sigma_F = 10^{-3}$	$\sigma_F = 10^{-3}$
	$\sigma_A = 10^{-2}$	$\sigma_A = 10^{-2}$	$\sigma_A = 10^{-1}$
	$\hat{\sigma}_A = 10^{-2}$	$\hat{\sigma}_A = 10^{-1}$	$\hat{\sigma}_A = 10^{-1}$
$P_A = 1$	$\sigma_F = 10^{-3}$	$\sigma_F = 10^{-4}$	$\sigma_F = 10^{-2}$
	$\sigma_A = 10^{-2}$	$\sigma_A = 10^{-2}$	$\sigma_A = 10^{-1}$
	$\hat{\sigma}_A = 10^{-1}$	$\hat{\sigma}_A = 10^{-1}$	$\hat{\sigma}_A = 10^{-1}$
$P_A = 10$	$\sigma_F = 10^{-4}$	$\sigma_F = 10^{-3}$	$\sigma_F = 10^{-3}$
	$\sigma_A = 10^{-2}$	$\sigma_A = 10^{-2}$	$\sigma_A = 10$
	$\hat{\sigma}_A = 10^{-2}$	$\hat{\sigma}_A = 10^{-1}$	$\hat{\sigma}_A = 10$

7.6 Flow Fingerprinting Game: correlated flows

We propose a second game, that is shown in Figure 7.8. In this game, the task of the TA is to add a fingerprint to the flow in such a way that he can differentiate between this flow and an identical flow that has not been fingerprinted. The goal of the adversary is to modify the fingerprinted flow in such a way that the TA decides that there is no fingerprint.

In this game, we also consider that the flows enter at a particular time in the system, so we denote by δ^l to the sequence that represents the time difference between

Table 7.12: Scale parameter at the SPE for different amount of chaff traffic for Scenario 2 ($n = 20$, $A_{max} = 250$ms and $W_{max} = 100$ms).

	u_A^*	$\underline{u}_A$	$\overline{u}_A$
No chaff	$\sigma_F = 10^{-4}$ $\sigma_A = 10^{-2}$ $\hat{\sigma}_A = 10^{-2}$	$\sigma_F = 10^{-3}$ $\sigma_A = 10^{-2}$ $\hat{\sigma}_A = 10^{-1}$	$\sigma_F = 10^{-3}$ $\sigma_A = 10^{-1}$ $\hat{\sigma}_A = 10^{-1}$
$P_A = 1$	$\sigma_F = 10^{-4}$ $\sigma_A = 10^{-2}$ $\hat{\sigma}_A = 10^{-2}$	$\sigma_F = 10^{-3}$ $\sigma_A = 10^{-2}$ $\hat{\sigma}_A = 10^{-1}$	$\sigma_F = 10^{-3}$ $\sigma_A = 10^{-1}$ $\hat{\sigma}_A = 10^{-1}$
$P_A = 10$	$\sigma_F = 10^{-2}$ $\sigma_A = 10^{-2}$ $\hat{\sigma}_A = 10^{-2}$	$\sigma_F = 10^{-3}$ $\sigma_A = 10^{-1}$ $\hat{\sigma}_A = 10^{-1}$	$\sigma_F = 10^{-4}$ $\sigma_A = 1$ $\hat{\sigma}_A = 1$

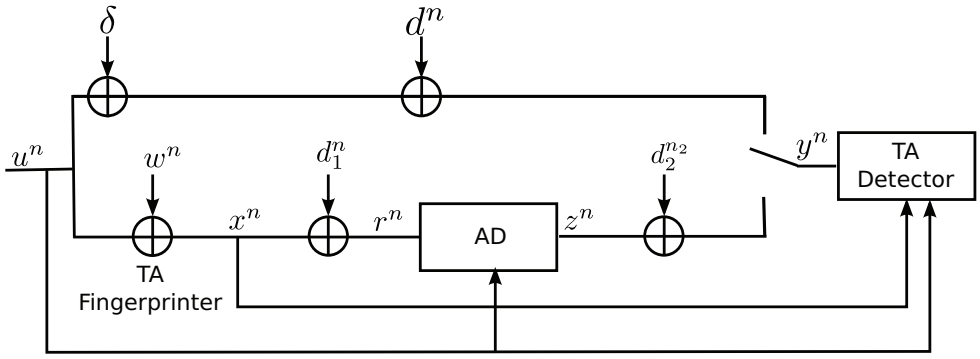


Figure 7.8: Model of the Flow Fingerprinting Game: correlated flows

the moment that the fingerprinted flow and an identical flow enter the system. The rest of the conditions are identical to the previous game. Recall that $D^n = D_1^n + D_2^n$ is the delay suffered by the flow and its distribution is identical for all the flows. A real scenario for this game may correspond to a web page trying to fingerprint a specific access so it can distinguish from other user's accesses to the same web page.

We also assume that the adversary can only delay packets, with the same constraint as in the previous game, i.e., $\forall_i \, a_i < A_{max}$. In this game, chaff traffic is not considered as it does not give any advantage to the adversary because the non-fingerprinted flow is known to have n packets, so any modification to the number of packets would identify it.

We assume that the first order statistics of D and ΔD are known by both players and define the hypotheses:

$$H_0 : y^n \text{ is a non-fingerprinted version of } u^n$$
$$H_1 : y^n \text{ is a fingerprinted version of } u^n.$$

We define the flow fingerprinting game for correlated flows as follows:

Definition 2. The $FFG_C(A_{TA}; A_{AD}; u)$ is a zero-sum game played by the TA and the adversary, where

- The set of actions the TA can choose from, A_{TA}, is the duple of possible fingerprint distributions f_{W^n} and acceptance regions Λ_1:

$$A_{TA} = \{f_{W^n} \times \Lambda_1 : 0 \leq w_i \leq W_{max},\ i \in [1, n]\} \tag{7.21}$$

- The adversary actions are the possible attack distributions with range in [0, A_{max}]:

$$A_{AD} = \{f_{A^n} : 0 \leq a_i \leq A_{max},\ i \in [1, n]\} \tag{7.22}$$

- We use two different utility functions, that are the same as previously.

7.6.1 Detector

In the detector for the uncorrelated game, we have assumed that both the TA and the attacker have only knowledge of the first order statistics of the PDV, ΔD. In this game, we assume that both players know the first order statistics of both the PDV and the delay, D. Hence, the optimal detector under this condition can be derived as

$$\begin{aligned}\Lambda_1(y^n, x^n, u^n, \hat{f}_{A^n|X^n}) =& \frac{\int_0^{A_{max}} f_D(y_1 - x_1 - a)\hat{f}_{A_1}(a)da}{\max_{\delta^L} f_D(y_1 - u_1 - \delta)} \\ & \cdot \prod_{i=1}^{n-1} \frac{\int_{\mathcal{R}} f_{\Delta D}(\Delta y_i - x_i + z))\hat{f}_{a_{i+1}-a_i|X^n}(z|x^n)dz}{f_{\Delta D}(\Delta y_i - u_i)}\end{aligned} \tag{7.23}$$

where $\hat{f}_{a_{i+1}-ai|X^n}(x) = \int_0^{A_{max}} \hat{f}_{A_{i+1}}(x+y)\hat{f}_{A_i}(y)dy$.

We assume that both the TA and the attacker do not know the sequence δ^L, i.e., they do not know when non-fingerprinted flows traverse the network. As the TA has to maximize over the range of possible δ, the term $\max_\delta f_D(y_1 - u_1 - \delta)$ becomes a constant equal to $\max_d f_D(d)$ that we can include in the threshold. Hence, the test becomes:

$$\Lambda_1(y^n, x^n, u^n, \hat{f}_{A^n|X^n}) = \int_0^{A_{max}} f_D(y_1 - x_1 - a)\hat{f}_{A_1}(a)da \cdot \prod_{i=1}^{n-1} \frac{\int_{\mathcal{R}} f_{\Delta D}(y_{i+1} - y_1 - x_{i+1} - x_i - z)\hat{f}_{a_{i+1}-a_i|X^n}(z|x^n)dz}{f_{\Delta D}(\Delta y_i - u_i)} \tag{7.24}$$

7.6.2 Deterministic attack

As in the previous game, we start studying the case when the attack and the fingerprint are deterministic. This means that they do not randomize with the aim of confusing the other player. Under these conditions $f_{A^n|R^n}(x) = \prod_{i=1}^n \delta(x - a_i)$ and $f_{\hat{A}^n|X^n}(x) = \prod_{i=1}^n \delta(x - \hat{a}_i)$. The test becomes:

$$\Lambda_1(y^n, x^n, u^n, \hat{a}^n) = f_D(y_1 - x_1 - \hat{a}_1) \prod_{i=1}^{n-1} \frac{f_{\Delta D}(\Delta_i(y^n - x^n - \hat{a}^n))}{f_{\Delta D}(\Delta(y^n - u^n))}. \tag{7.25}$$

The solutions of this game are:

$$u^* = \max_{w^n} \min_{a^n} Pr\left(f_D(d_1) \cdot \prod_{i=1}^{n-1} \frac{f_{\Delta D}(\Delta_i d^n)}{f_{\Delta D}(\Delta_i(w^n + a^n + d^n))} > \epsilon \right) \tag{7.26}$$

$$\underline{u} = \max_{w^n, \hat{a}^n} \min_{a^n} Pr\left(f_D(d_1 + a_1 - \hat{a}_1) \cdot \prod_{i=1}^{n-1} \frac{f_{\Delta D}(\Delta_i(d^n + a^n - \hat{a}^n))}{f_{\Delta D}(\Delta_i(w^n + a^n + d^n))} > \epsilon \right) \tag{7.27}$$

$$\overline{u} = \min_{a^n} \max_{w^n} Pr\left(f_D(d_1) \cdot \prod_{i=1}^{n-1} \frac{f_{\Delta D}(\Delta_i d^n)}{f_{\Delta D}(\Delta_i(w^n + a^n + d^n))} > \epsilon \right) \tag{7.28}$$

These solutions are obtained from equations (7.1),(7.2) and (7.3) using the test (7.25).

Note that a^n seems to be chosen to minimize $\Delta(a^n + w^n)$, hence destroying the watermark information in the IPD. This can be completely achieved when $A_{max} \geq W_{max}$, making the timing of the first packet the only available information for the test.

Results

We keep the same two scenarios for simulation. As in this game results depend on the time when identical flows enter the system, i.e., δ^l, we simulate those instants using the NASA 's and World Cup's logs presented in Chapter 3. Recall from Table 3.3 that the NASA web server received an average of 0.66 requests per second and the World Cup web server 14.97 requests per second.

We simulate the following situations:

1. No watermark
2. Watermark without attack
3. Watermark with attack $A_{max} < W_{max}$
4. Watermark with attack $A_{max} = W_{max}$
5. No attack, but watermark amplitude as $W_{max} - A_{max}$ of the third case.

In this game, we do not consider $\overline{u}$ since the attack cannot be chosen before the fingerprint value.

Results are depicted in Figures 7.10 and 7.9 for Scenario 1 (Amazon cloud) and in Figures 7.12 and 7.11 for Scenario 2 (Tor network). Recall that Scenario 2 has a much larger delay variance and also its PDV is much more dispersed.

First, we notice that even when no fingerprint is added the fact that δ takes specific values allows the TA to guess with some confidence when $E[\Delta\delta]$ is smaller

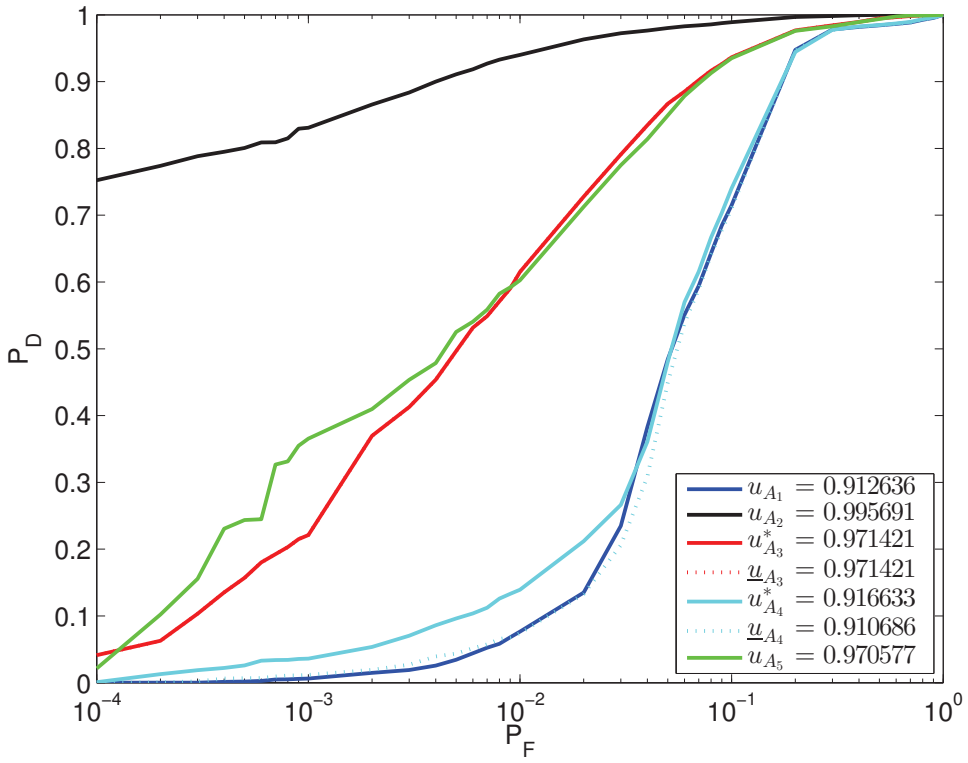


Figure 7.9: Solution of the constant games for Scenario 1 with World Cup log and $n = 5$, with game parameters: $A_1 : (A_{max} = 0,\ F_{max} = 0)$, $A_2 : (A_{max} = 0, F_{max} = 1ms)$, $A_3 : (A_{max} = 0.75ms,\ F_{max} = 1ms)$, $A_4 : (A_{max} = 1ms,\ F_{max} = 1ms)$ and $A_5 : (A_{max} = 0ms,\ F_{max} = 0.25ms)$.

than the standard deviation of the network delay. Second, when $A_{max} \geq W_{max}$ the attacker will destroy the watermark and the results are very similar to the non-fingerprint situation. In Scenario 1, the presence of the attacker makes the performance very similar to a fingerprint of amplitude $W_{max} - A_{max}$ but in Scenario 2 the attacker impairs the correlation performance more severely than what would be obtained with no attack and a fingerprint amplitude of $W_{max} - A_{max}$.

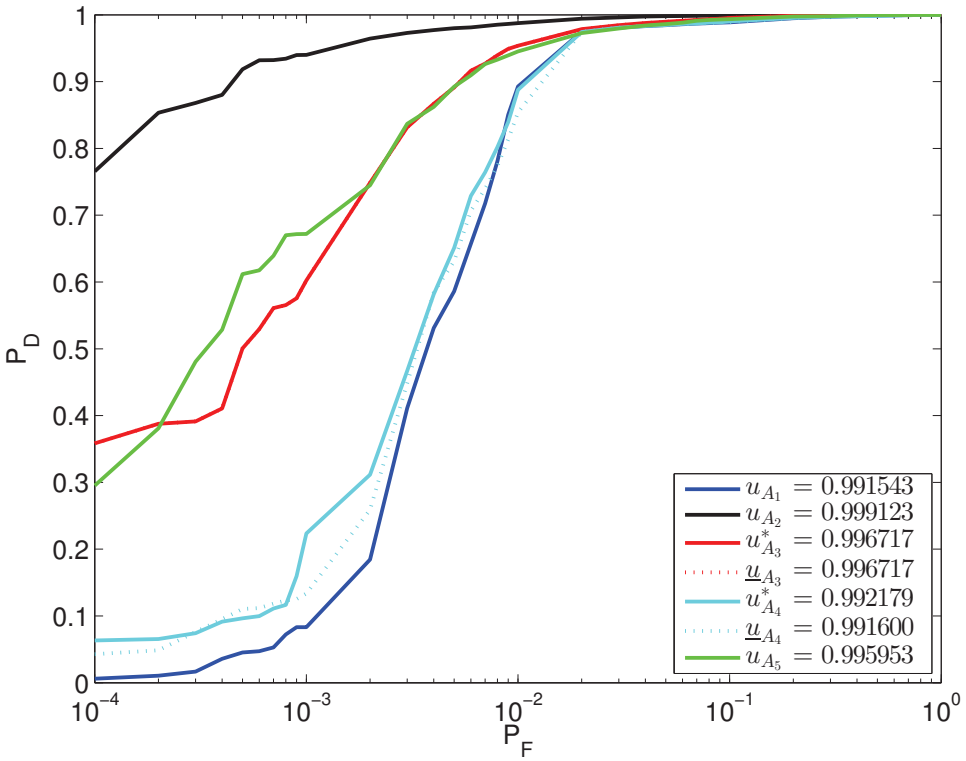


Figure 7.10: Solution of the constant games for Scenario 1 with NASA's log and $n = 5$, with game parameters: $A_1 : (A_{max} = 0,\ F_{max} = 0)$, $A_2 : (A_{max} = 0, F_{max} = 1ms)$, $A_3 : (A_{max} = 0.75ms,\ F_{max} = 1ms)$, $A_4 : (A_{max} = 1ms,\ F_{max} = 1ms)$ and $A_5 : (A_{max} = 0ms,\ F_{max} = 0.25ms)$.

7.6.3 Truncated-Gaussian Attack

As in the previous game, we study the case when the attack is randomized using a truncated Gaussian distribution. Afterwards, in the next section, we compare the results with the other distribution attacks used previously, namely the truncated Laplacian and the truncated Cauchy distributions. Finally, we study the case of the attacker chooses a different model than that assumed by the test.

As mentioned, the adversary add delays following a truncated Gaussian in the allowed interval so as the TA cannot guess its value and compensates it in the test. Therefore, $a_i \sim N(\mu_{a,i}, \sigma_a^2 | 0 \leq a_i \leq A_{max})$. Recall that this game includes

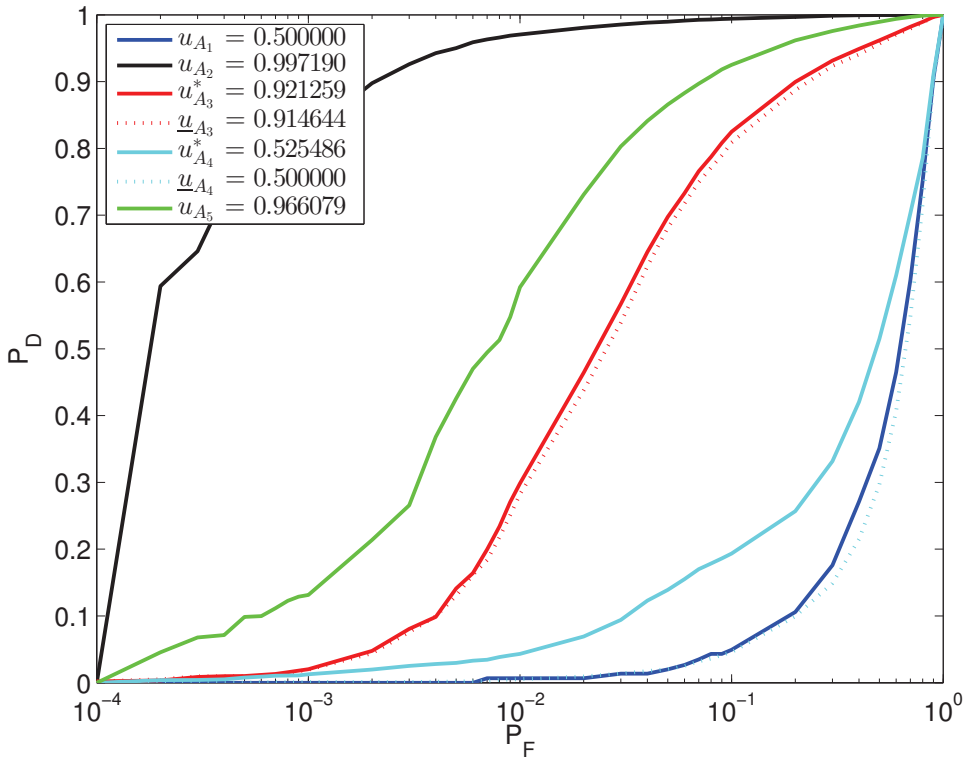


Figure 7.11: Solution of the constant games for Scenario 2 with World Cup's log and $n = 10$, with game parameters: $A_1 : (A_{max} = 0,\ F_{max} = 0)$, $A_2 : (A_{max} = 0, F_{max} = 100ms)$, $A_3 : (A_{max} = 75ms,\ F_{max} = 100ms)$, $A_4 : (A_{max} = 100ms,\ F_{max} = 100ms)$ and $A_5 : (A_{max} = 0ms,\ F_{max} = 25ms)$.

as extremes cases the deterministic attack seen previously (i.e., $\sigma^2 \to 0$), and a uniform attack (i.e., $\sigma^2 \to \infty$). In this game, i.e., correlated flows, the TA does not achieve any advantage by randomizing the fingerprint as the attacker knows u^n, so the fingerprint is still created in a deterministic way.

Formally, the actions available to each player are:

$$A_{TA} = \{w^n \times \hat{\mu}^n \times \hat{\sigma}^2\} \tag{7.29}$$

$$A_{AD} = \{\mu^n \times \sigma^2\} \tag{7.30}$$

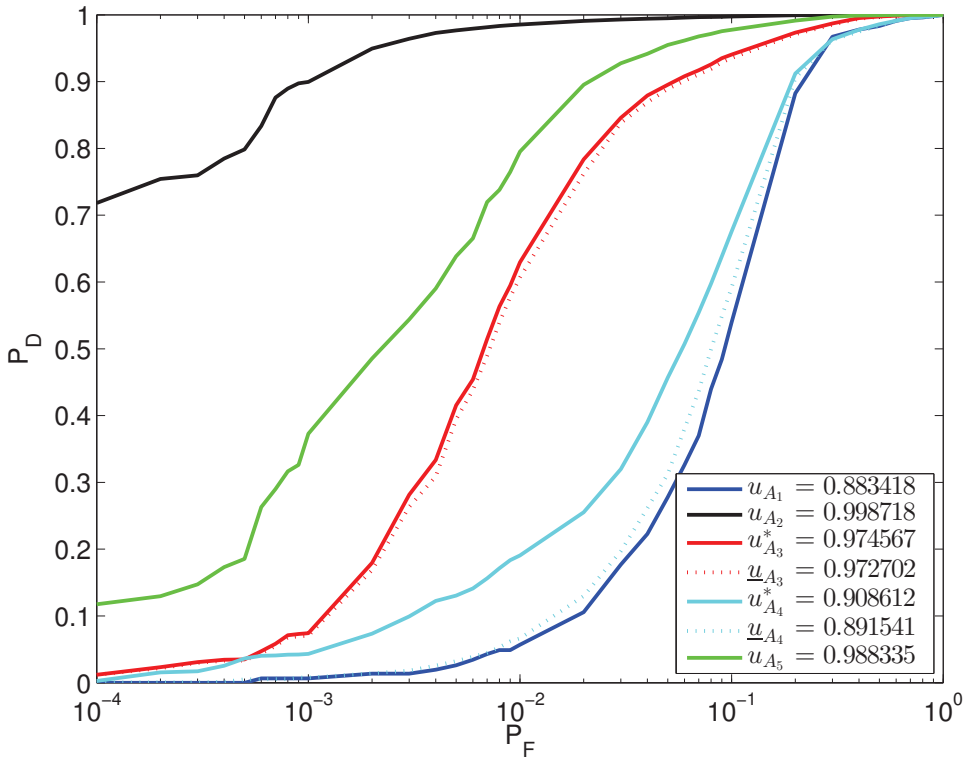


Figure 7.12: Solution of the constant games for Scenario 2 with NASA's log and $n = 10$, with game parameters: $A_1 : (A_{max} = 0,\ F_{max} = 0)$, $A_2 : (A_{max} = 0, F_{max} = 100ms)$, $A_3 : (A_{max} = 75ms,\ F_{max} = 100ms)$, $A_4 : (A_{max} = 100ms,\ F_{max} = 100ms)$ and $A_5 : (A_{max} = 0ms,\ F_{max} = 25ms)$.

and the detector is:

$$\Lambda_1(y^n, x^n, u^n, \hat{a}^n, \hat{\sigma}^2) = \int_0^{A_{max}} f_D(y_1 - x_1 - a) f_{TG}(a|\hat{a_1}, \hat{\sigma}^2) da \cdot \prod_{i=1}^{n-1} \frac{\int_{\mathcal{R}} f_{\Delta D}(y_{i+1} - y_1 - x_{i+1} - x_i - z) \hat{f}_{DTG}(z|\hat{a}_{i+1}, \hat{a}_i, \hat{\sigma}^2) dz}{f_{\Delta D}(\Delta y_i - u_i)} \tag{7.31}$$

with

$$f_{TG}(x|a, \sigma^2) = \frac{\phi\left(\frac{x-a}{\sigma}\right)}{\sigma\left(\Phi\left(\frac{A_{max}-a}{\sigma}\right) - \Phi\left(\frac{-a}{\sigma}\right)\right)} \tag{7.32}$$

and

$$f_{DTG}(x|a_2, a_1, \sigma^2) == \int_0^{A_{max}} f_{TG}(y|a_2, \sigma^2) \hat{f}_{TG}(y - z|a_1, \sigma^2) dy \tag{7.33}$$

Table 7.13: $u_A^*(\sigma_a)$ for Scenario 1 with WC's log ($n = 5$, $A_{max} = 0.75$ms and $W_{max} = 1$ms).

σ_A/A_{max}	10^{-4}	10^{-3}	10^{-2}	10^{-1}	1	10
	0.9833	0.9834	0.9843	0.9909	0.9977	0.9979

Table 7.14: $\underline{u}_A(\sigma_a, \hat{\sigma}_a)$ for Scenario 1 ($n = 5$, $A_{max} = 0.75$ms and $W_{max} = 1$ ms).

$\hat{\sigma}/A_{max}$ \ σ/A_{max}	10^{-4}	10^{-3}	10^{-2}	10^{-1}	1	10
10^{-4}	0.9833	0.9833	0.9841	0.9881	0.9842	0.9834
10^{-3}	0.9833	0.9834	0.9841	0.9883	0.9843	0.9831
10^{-2}	0.9835	0.9836	0.9843	0.9886	0.9853	0.9842
10^{-1}	**0.9841**	0.9842	0.9853	0.9908	0.9915	0.9908
1	0.9708	0.9710	0.9731	0.9861	0.9978	0.9979
10	0.9696	0.9697	0.9714	0.9852	0.9978	0.9980

being ϕ and Φ respectively the pdf and the cdf of a standard normal distribution.

We simulate this game using the A_4 conditions of the previous section, i.e. $0 < A_{max} < W_{max}$, with World Cup's log (WC) for Scenario 1 and NASA's log for Scenario 2. Results of u^* and $\underline{u}$ are provided in Tables 7.13 and 7.14 for Scenario 1 and in Tables 7.15 and 7.16 for Scenario 2. We can see that the attacker chooses $\sigma \in [10^{-4}, 10^{-3}] \cdot A_{max}$, this small variance makes the attack virtually deterministic as in the previous game. When the detector parameters are chosen without knowing the attack ones, i.e. $\underline{u}$, in a low variance delay, (Scenario 1), the TA has to overestimate the attack variance, $10^{-1} \cdot A_{max}$ instead of $[10^{-4}, 10^{-3}] \cdot A_{max}$, so the adversary does not have the chance to choose a higher variance attack to degrade the detector performance. This does not happen in Scenario 2, as even though the detector is known to consider a low variance attack the adversary's best option is to use a small variance, due to the high variance distributions of D and ΔD.

Table 7.15: $u_A^*(\sigma_a)$ for Scenario 2 with NASA's log ($n = 10$, $A_{max} = 75$ms and $W_{max} = 100$ms).

σ_A/A_{max}	10^{-4}	10^{-3}	10^{-2}	10^{-1}	1	10
	0.9930	**0.9920**	0.9932	0.9945	0.9977	0.9981

Table 7.16: $\underline{u}_A(\sigma_a, \hat{\sigma}_a)$ for Scenario 2 with NASA's log($n = 10$, $A_{max} = 75$ms and $W_{max} = 100$ ms).

$\hat{\sigma}/A_{max}$ \ σ/A_{max}	10^{-4}	10^{-3}	10^{-2}	10^{-1}	1	10
10^{-4}	**0.9920**	0.9921	0.9925	0.9942	0.9939	0.9939
10^{-3}	0.9908	0.9908	0.9913	0.9938	0.9942	0.9941
10^{-2}	0.9919	0.9920	0.9924	0.9943	0.9946	0.9944
10^{-1}	0.9894	0.9895	0.9902	0.9943	0.9968	0.9969
1	0.9644	0.9646	0.9669	0.9838	0.9977	0.9980
10	0.9609	0.9612	0.9638	0.9821	0.9978	0.9981

7.6.4 Other distribution attacks

As in the previous game, we want to analyze if there are significant differences when instead of choosing a truncated Gaussian the adversary chooses different distributions. We use the Laplace and Cauchy distributions truncated to $[0, A_{max}]$. Their pdfs are shown in Table 6.3. We solve the game identically to the previous section and we show the utility and the scale parameter at the SPE for the proposed distributions in Tables 7.17 and 7.18, where we can see that the adversary impact does not vary significantly depending on the distribution it follows. This fact can be better seen in Figures 7.13 and 7.14 where we depict the ROC curve at the SPE. We see that in Scenario 1 the adversary should choose a Gaussian model and in Scenario 2 even the AUC is smaller for a Laplace model, the figure shows that the Gaussian model gives worse performance for $P_F < 10^{-2}$.

Table 7.17: Comparison of the utility at the SPE for different distribution attacks and the scale parameters at the SPE in Scenario 1 with WC log ($n = 5$, $A_{max} = 0.75$ms and $W_{max} = 1$ms).

	u_A^*	$\sigma*$	$\underline{u}_A$	$\underline{\sigma}$, $\underline{\hat{\sigma}}$
Truncated Gaussian	0.9810	10^{-4}	0.9817	10^{-4}, 10^{-1}
Truncated Laplace	0.9856	10^{-4}	0.9850	10^{-4}, 10^{-2}
Truncated Cauchy	0.9840	10^{-4}	0.9839	10^{-4}, 10^{-2}

Table 7.18: Comparison of the utility at the SPE for different distribution attacks and the scale parameters at the SPE in Scenario 2 with NASA's log ($n = 10$, $A_{max} = 75$ms and $W_{max} = 100$ms).

	u_A^*	$\sigma*$	$\underline{u}_A$	$\underline{\sigma}$, $\hat{\underline{\sigma}}$
Truncated Gaussian	0.9928	10^{-3}	0.9926	10^{-4}, 10^{-4}
Truncated Laplace	0.9915	10^{-3}	0.9904	10^{-4}, 10^{-4}
Truncated Cauchy	0.9939	10^{-3}	0.9939	10^{-4}, 10^{-2}

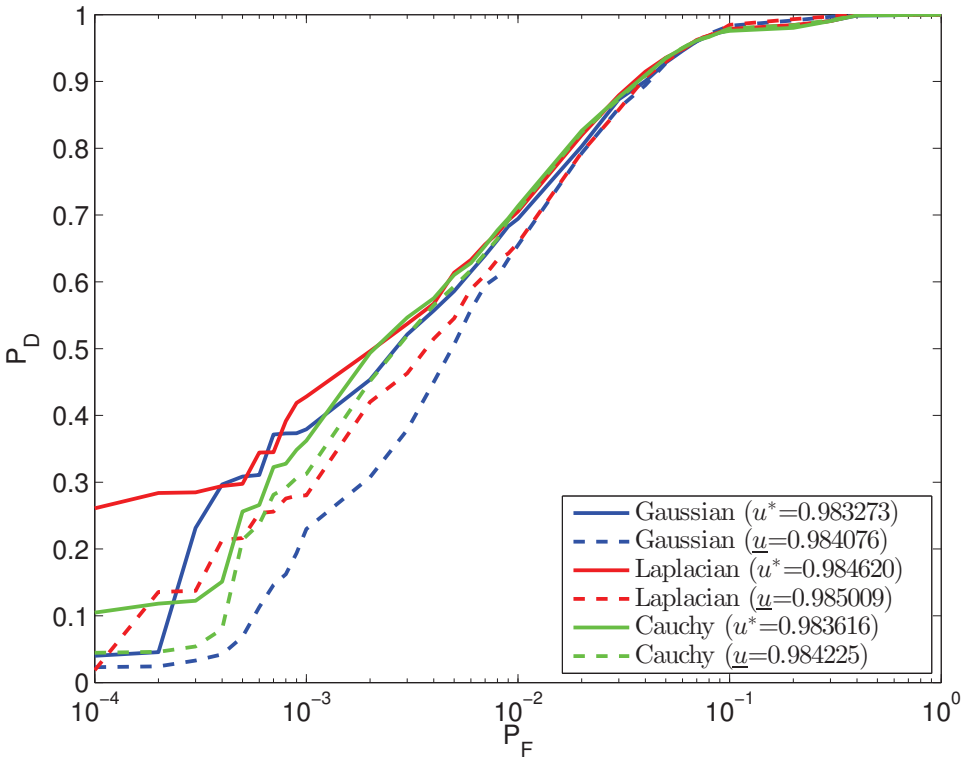


Figure 7.13: Solution of the truncated Gaussian (blue), truncated Laplace (red) and truncated Cauchy (green) correlated flow games for Scenario 1 with WC's log($n = 5$, $A_{max} = 0.75$ms and $W_{max} = 1$ms).

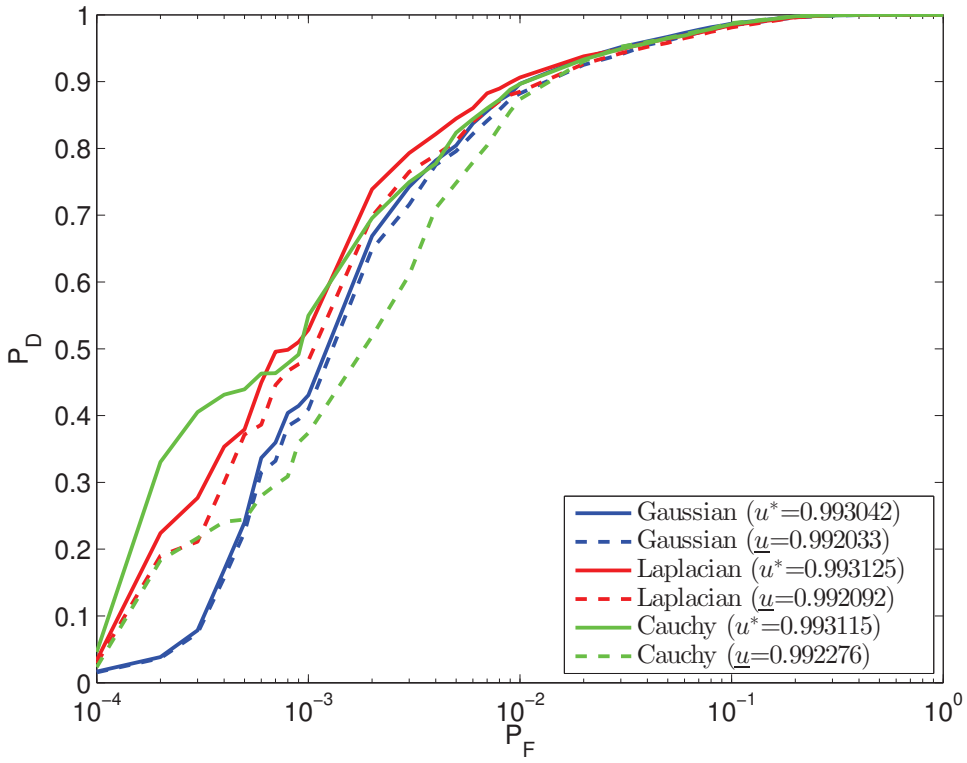


Figure 7.14: Solution of the truncated Gaussian (blue), truncated Laplace (red) and truncated Cauchy (green) correlated flow games for Scenario 2 with NASA's log($n = 10$, $A_{max} = 75$ms and $W_{max} = 100$ms).

7.6.5 Distribution mismatch between the adversary and the decoder

Finally, we study the consequences of the adversary choosing a different attack distribution than that the assumed by the detector. This situation only is possible in $\underline{u}$.

We assume a detector that assumes a truncated Gaussian attack as according to the previous section the adversary should choose this model as it gives slightly smaller utility. Given that the adversary knows this fact, we want to study whether the adversary can choose a different distribution attack to impair the test in a more severe

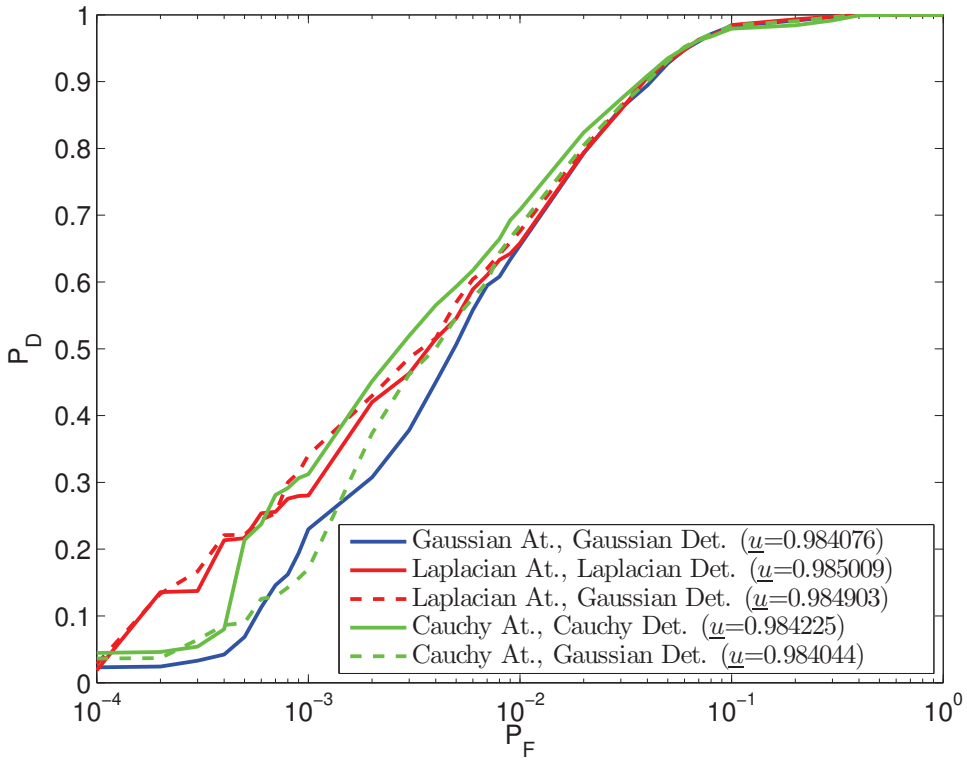


Figure 7.15: ROC curves for Gaussian detector ($\hat{\sigma} = 10^{-1}$) for Scenario 1 with WC's $\log(n = 5,\ A_{max} = 0.75\text{ms}$ and $W_{max} = 1\text{ms})$.

way. We assume that the adversary can choose among the following distributions: truncated Gaussian, truncated Laplacian and truncated Cauchy.

We simulate the same scenarios with the same parameters as in the previous section, and we depict the results in Figures 7.15 and 7.16 for Scenarios 1 and 2, respectively. We can see that even the TA is known to make a truncated Gaussian assumption the adversary does not improve its utility significantly by selecting a different distribution attack.

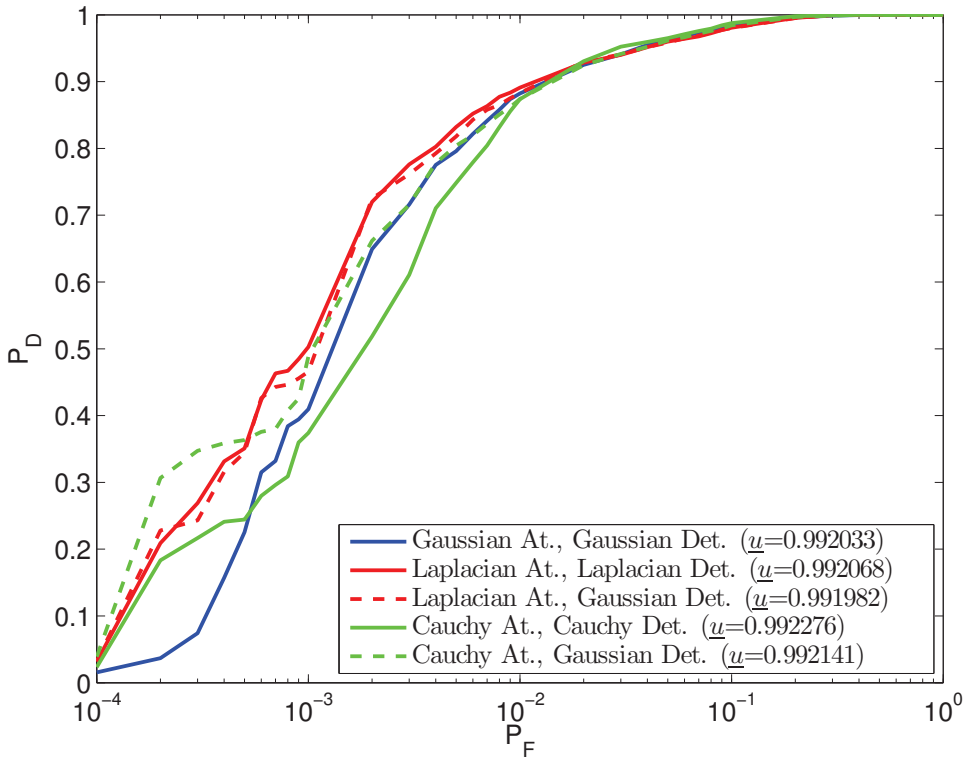


Figure 7.16: ROC curves for Gaussian detector ($\hat{\sigma} = 10^{-4}$) for Scenario 2 with NASA's log($n = 10$, $A_{max} = 75$ms and $W_{max} = 100$ms).

7.7 Conclusion

In this chapter, we have analyzed two different flow fingerprinting games. In the first one, the TA's goal is to differentiate between a known flow or different flows where the fingerprint is added to improve the performance. In the second game, the goal of the TA is to differentiate between a fingerprinted flow and identical flows that have not been fingerprinted. In both games, we assume a rational adversary who tries to impair the correlation as much as possible. Results show that for both games, the TA and the attacker should act in a virtually deterministic way in most of the cases.

Chapter 8

Conclusions and Future Work

This section briefly summarizes the conclusions that may be extracted from the research work undertaken in the present thesis.

We started by proposing a method to leave a fingerprint in a log that stores the timing of certain events that the fingerprinter can create. The trace is hidden in the timing information in order to make it hard to be detected by the log owner or any other party that does not have the creation timing. We applied this technique to fingerprint Tor's hidden web servers. We also showed that an optimal decoder, i.e., a likelihood test, may not be the best solution for all the scenarios, as in some circumstances a suboptimal decoder can achieve similar results, being much easier to compute and calculate the theoretical performance.

We have proposed two methods for the forensic problem of tracing flows that are involved in low-latency anonymous networks. These methods are designed to handle a large amount of dummy or other users' traffic without being able to differentiate them. We use both flow correlation methods to locate Tor's hidden services, but the ideas behind these methods can be used in many other contexts such as deciding whether a flow goes through a certain proxy, tracing VPN flows, etc.

Afterwards, we addressed the classical flow-correlation problem, showing that a passive analysis technique can outperform the state-of-the-art watermarking techniques, in the case of non-highly-correlated flows. This is achieved by using an optimal decoder that includes statistical modelling. This method is designed to be robust against an adversary that adds chaff traffic, splits the flows or adds random delays. We also showed the performance of the non-blind watermark extension of this algorithm.

Finally, we studied the limits of flow watermarking/fingerprinting under an adversary that tries to destroy it. We showed that in most of the possible scenarios the TA and the attacker should act in a virtually deterministic way making the flow the most or least unique as possible, respectively, instead of confusing the other player. We also showed that we need a huge amount of dummy packets in order to impair the correlation significantly.

8.1 Future Research Lines

Traffic analysis in IP networks has many open hot topics that will be progressively tackled in the near future. The ones most directly related to the research covered in this thesis are briey highlighted in the following points:

1. A perfect invisible watermark against an attacker that knows only the first-order distribution of the PDVs, i.e., ΔD, is feasible. Creating this watermark and studying the improvement of performance against a passive analysis is a problem to address. This watermark has to be the optimal watermark among those which keep $D_{KL}(\Delta D||\Delta D + \Delta W) = 0$.

2. Creating a flow fingerprint is indeed the same problem as creating a covert channel where the information is sent on the timing of packets. Analyzing the

trade-off of invisibility against the rate of information that can be reliably sent is an important open problem that has not been addressed yet.

References

[1] W. Friedman and L. Callimahos, *Military Cryptanalytics*, ser. Cryptographic series. Aegean Park Press, 1985, no. v. 1.

[2] D. A. Borrmann, W. T. Kvetkas, C. V. Brown, M. J. Flatley, and R. Hunt, "The history of traffic analysis: World War I - Vietnam," 2013.

[3] W. Friedman, L. Callimahos, and W. Barker, *Military cryptanalytics*, ser. Cryptographic series. Aegean Park Press, 1985, no. pt. 1, v. 2; pt. 2, v. 1.

[4] Imperial War Museum Collections Photograph.

[5] E. Layton, R. Pineau, and J. Costello, *And I Was There: Pearl Harbor and Midway–Breaking the Secrets*. Morrow, 1985.

[6] P. Beesly, J. Rohwer, and K. Knowles, "Ultra and the Battle of the Atlantic," *Cryptologic Spectrum*, vol. 8, no. 1, 1978, declasiffied: July 2010.

[7] R. Lewin, *THE AMERICAN MAGIC: Codes, Cyphers and the Defeat of Japan*. Farrar Straus & Giroux, 1982.

[8] D. Kahn, *Hitler's spies : German military intelligence in World War II*. Cambridge, Mass: Da Capo Press, 2000.

[9] R. Kusaka, *Rengo Kantai (Combined Fleet)*, ser. The Pearl Harbor Papers: Inside The Japanese Plans, D. M. Goldstein and K. V. Dillon, Eds. Brasseys, 1993.

[10] A. Norman, *Operation Overlord, design and reality; the Allied invasion of Western Europe*. Westport, Conn: Greenwood Press, 1970.

[11] National Security Agency, "The Origination and Evolution of Radio Traffic Analysis: The World War I Era," *Cryptologic Quarterly*, vol. 6, no. 1, pp. 21–40, Spring 1987.

[12] ——, "The Origination and Evolution of Radio Traffic Analysis: The Period between the Wars," *Cryptologic Quarterly*, vol. 6, no. 3-4, pp. 21–40, Fall/Winter 1987-1988.

[13] J. T. Richelson, *A Century of Spies : Intelligence in the Twentieth Century: Intelligence in the Twentieth Century.* Oxford University Press, USA, 1995.

[14] W. Grayson, *Chicksands, a Millennium of History.* Shefford Press, 1994.

[15] M. Herman and R. I. of International Affairs, *Intelligence Power in Peace and War.* Cambridge University Press, 1996.

[16] S. Gorman and J. Valentino-Devries, "New details show broader nsa surveillance reach," *The Wall Street Journal*, Aug. 20, 2013.

[17] D. X. Song, D. Wagner, and X. Tian, "Timing analysis of keystrokes and timing attacks on ssh," in *10th USENIX Security Symposium*, vol. 2, 2001, p. 3.

[18] F. Monrose, M. K. Reiter, and S. Wetzel, "Password hardening based on keystroke dynamics," *International Journal of Information Security*, vol. 1, no. 2, pp. 69–83, 2002.

[19] D. Asonov and R. Agrawal, "Keyboard acoustic emanations," in *Security and Privacy, 2004. Proceedings. 2004 IEEE Symposium on.* IEEE, 2004, pp. 3–11.

[20] L. Zhuang, F. Zhou, and J. D. Tygar, "Keyboard acoustic emanations revisited," in *Proceedings of the 12th ACM conference on Computer and communications security.* ACM, 2005, pp. 373–382.

[21] M. Vuagnoux and S. Pasini, "Compromising electromagnetic emanations of wired and wireless keyboards." in *USENIX Security Symposium*, 2009, pp. 1–16.

[22] K. Zhang and X. Wang, "Peeping Tom in the neighborhood: keystroke eavesdropping on multi-user systems," *analysis*, vol. 20, p. 23, 2009.

[23] P. Marquardt, A. Verma, H. Carter, and P. Traynor, "(sp) iphone: decoding vibrations from nearby keyboards using mobile phone accelerometers," in *Proceedings of the 18th ACM conference on Computer and communications security.* ACM, 2011, pp. 551–562.

[24] L. Cai and H. Chen, "Touchlogger: inferring keystrokes on touch screen from smartphone motion," in *Proceedings of the 6th USENIX conference on Hot topics in security.* USENIX Association, 2011, pp. 9–9.

[25] T. Ristenpart, E. Tromer, H. Shacham, and S. Savage, "Hey, you, get off of my cloud: exploring information leakage in third-party compute clouds," in *Proceedings of the 16th ACM conference on Computer and communications security*. ACM, 2009, pp. 199–212.

[26] S. Mistry and B. Raman, "Quantifying traffic analysis of encrypted web-browsing." December 1998.

[27] H. Cheng and R. Avnur, ""traffic analysis of ssl encrypted web browsing"," http://www.cs.berkeley.edu/~daw/teaching/cs261-f98/projects/final-reports/ronathan-heyning.ps, 1998.

[28] A. Hintz, "Fingerprinting websites using traffic analysis," in *Privacy Enhancing Technologies*. Springer, 2003, pp. 229–233.

[29] Q. Sun, D. R. Simon, Y.-M. Wang, W. Russell, V. N. Padmanabhan, and L. Qiu, "Statistical identification of encrypted web browsing traffic," in *Security and Privacy, 2002. Proceedings. 2002 IEEE Symposium on*. IEEE, 2002, pp. 19–30.

[30] G. Bissias, M. Liberatore, D. Jensen, and B. Levine, "Privacy vulnerabilities in encrypted http streams," in *Privacy Enhancing Technologies*. Springer, 2006, pp. 1–11.

[31] M. Liberatore and B. N. Levine, "Inferring the source of encrypted http connections," in *Proceedings of the 13th ACM conference on Computer and communications security*. ACM, 2006, pp. 255–263.

[32] D. Herrmann, R. Wendolsky, and H. Federrath, "Website fingerprinting: attacking popular privacy enhancing technologies with the multinomial naïve-bayes classifier," in *Proceedings of the 2009 ACM workshop on Cloud computing security*. ACM, 2009, pp. 31–42.

[33] R. Dingledine, N. Mathewson, and P. Syverson, "Tor: the second-generation onion router," in *Proceedings of the 13th conference on USENIX Security Symposium - Volume 13*, ser. SSYM'04. Berkeley, CA, USA: USENIX Association, 2004, pp. 21–21.

[34] O. Berthold, H. Federrath, and M. Kóhntopp, "Project anonymity and unobservability in the internet," in *Proceedings of the tenth conference on Computers, freedom and privacy: challenging the assumptions*, ser. CFP '00. New York, NY, USA: ACM, 2000, pp. 57–65.

[35] A. Panchenko, L. Niessen, A. Zinnen, and T. Engel, "Website fingerprinting in onion routing based anonymization networks," in *Proceedings of the 10th*

annual ACM workshop on Privacy in the electronic society. ACM, 2011, pp. 103–114.

[36] S. E. Coull, M. P. Collins, C. V. Wright, F. Monrose, M. K. Reiter *et al.*, "On web browsing privacy in anonymized netflows," in *Proceedings of the 16th USENIX Security Symposium*, 2007, pp. 339–352.

[37] S. Chen, R. Wang, X. Wang, and K. Zhang, "Side-channel leaks in web applications: A reality today, a challenge tomorrow," in *Security and Privacy (SP), 2010 IEEE Symposium on*. IEEE, 2010, pp. 191–206.

[38] E. W. Felten and M. A. Schneider, "Timing attacks on web privacy," in *Proceedings of the 7th ACM conference on Computer and communications security*. ACM, 2000, pp. 25–32.

[39] C. Jackson, A. Bortz, D. Boneh, and J. C. Mitchell, "Protecting browser state from web privacy attacks," in *Proceedings of the 15th international conference on World Wide Web*, ser. WWW '06. New York, NY, USA: ACM, 2006, pp. 737–744.

[40] D. Jang, R. Jhala, S. Lerner, and H. Shacham, "An empirical study of privacy-violating information flows in javascript web applications," in *Proceedings of the 17th ACM conference on Computer and communications security*, ser. CCS '10. New York, NY, USA: ACM, 2010, pp. 270–283.

[41] T. Kohno, A. Broido, and K. Claffy, "Remote physical device fingerprinting," *IEEE Transactions on Dependable and Secure Computing*, vol. 2, no. 2, pp. 93–108, May 2005.

[42] S. J. Murdoch, "Hot or not: Revealing hidden services by their clock skew," in *Proceedings of the 13th ACM conference on Computer and communications security*. ACM, 2006, pp. 27–36.

[43] S. Zander and S. J. Murdoch, "An improved clock-skew measurement technique for revealing hidden services." in *USENIX Security Symposium*, 2008, pp. 211–226.

[44] S. M. Bellovin, "A technique for counting natted hosts," in *Proceedings of the 2nd ACM SIGCOMM Workshop on Internet measurment*. ACM, 2002, pp. 267–272.

[45] R. Beverly, "A robust classifier for passive tcp/ip fingerprinting," in *Passive and Active Network Measurement*. Springer, 2004, pp. 158–167.

[46] J. Franklin, D. McCoy, P. Tabriz, V. Neagoe, J. V. Randwyk, and D. Sicker, "Passive data link layer 802.11 wireless device driver fingerprinting," in *Proc. 15th USENIX Security Symposium*, 2006, pp. 167–178.

[47] D. Brockmann, L. Hufnagel, and T. Geisel, "The scaling laws of human travel," *Nature*, vol. 439, no. 7075, pp. 462–465, 2006.

[48] M. C. Gonzalez, C. A. Hidalgo, and A.-L. Barabasi, "Understanding individual human mobility patterns," *Nature*, vol. 453, no. 7196, pp. 779–782, 2008.

[49] E. Cho, S. A. Myers, and J. Leskovec, "Friendship and mobility: user movement in location-based social networks," in *Proceedings of the 17th ACM SIGKDD international conference on Knowledge discovery and data mining*, ser. KDD '11. New York, NY, USA: ACM, 2011, pp. 1082–1090.

[50] C. Song, Z. Qu, N. Blumm, and A.-L. Barabási, "Limits of predictability in human mobility," *Science*, vol. 327, no. 5968, pp. 1018–1021, 2010.

[51] Y.-A. de Montjoye, C. A. Hidalgo, M. Verleysen, and V. D. Blondel, "Unique in the crowd: The privacy bounds of human mobility," *Scientific reports*, vol. 3, 2013.

[52] M. Kim, D. Kotz, and S. Kim, "Extracting a mobility model from real user traces." in *INFOCOM*, vol. 6, 2006, pp. 1–13.

[53] A. J. Nicholson and B. D. Noble, "Breadcrumbs: forecasting mobile connectivity," in *Proceedings of the 14th ACM international conference on Mobile computing and networking*. ACM, 2008, pp. 46–57.

[54] N. Eagle, A. S. Pentland, and D. Lazer, "Inferring friendship network structure by using mobile phone data," *Proceedings of the National Academy of Sciences*, vol. 106, no. 36, pp. 15 274–15 278, 2009.

[55] L. Humphreys, "Mobile social networks and social practice: A case study of dodgeball," *Journal of Computer-Mediated Communication*, vol. 13, no. 1, pp. 341–360, 2007.

[56] A. Noulas, S. Scellato, C. Mascolo, and M. Pontil, "An empirical study of geographic user activity patterns in foursquare." *ICWSM*, vol. 11, pp. 70–573, 2011.

[57] S. Scellato, A. Noulas, R. Lambiotte, and C. Mascolo, "Socio-spatial properties of online location-based social networks." *ICWSM*, vol. 11, pp. 329–336, 2011.

[58] A. Noulas, S. Scellato, R. Lambiotte, M. Pontil, and C. Mascolo, "A tale of many cities: universal patterns in human urban mobility," *PloS one*, vol. 7, no. 5, p. e37027, 2012.

[59] A. Sadilek, H. Kautz, and J. P. Bigham, "Finding your friends and following them to where you are," in *Proceedings of the fifth ACM international conference on Web search and data mining*, ser. WSDM '12. New York, NY, USA: ACM, 2012, pp. 723 – 732.

[60] D. L. Chaum, "Untraceable electronic mail, return addresses, and digital pseudonyms," *Communications of the ACM*, vol. 24, no. 2, pp. 84–90, 1981.

[61] S. Parekh, "Prospects for remailers," *First Monday*, vol. 1, no. 2, 1996.

[62] L. Cottrell, "Mixmaster & remailer attacks, 1995," *http://www.obscura.com/~loki/remailer/remailer-essay.html*.

[63] G. Danezis, R. Dingledine, and N. Mathewson, "Mixminion: Design of a type iii anonymous remailer protocol," in *Security and Privacy, 2003. Proceedings. 2003 Symposium on*. IEEE, 2003, pp. 2–15.

[64] D. M. Goldschlag, M. G. Reed, and P. F. Syverson, "Hiding routing information," in *Information Hiding*. Springer, 1996, pp. 137–150.

[65] M. G. Reed, P. F. Syverson, and D. M. Goldschlag, "Anonymous connections and onion routing," *Selected Areas in Communications, IEEE Journal on*, vol. 16, no. 4, pp. 482–494, 1998.

[66] D. Goldschlag, M. Reed, and P. Syverson, "Onion routing," *Communications of the ACM*, vol. 42, no. 2, pp. 39–41, 1999.

[67] P. Syverson, G. Tsudik, M. Reed, and C. Landwehr, "Towards an analysis of onion routing security," in *Designing Privacy Enhancing Technologies*. Springer, 2001, pp. 96–114.

[68] J.-F. Raymond, "Traffic analysis: Protocols, attacks, design issues, and open problems," in *Designing Privacy Enhancing Technologies*. Springer, 2001, pp. 10–29.

[69] P. Boucher, A. Shostack, and I. Goldberg, "Freedom systems 2.0 architecture. white paper, zero knowledge systems," *White Paper, Zero Knowledge Systems, Inc.*

[70] A. Back, I. Goldberg, and A. Shostack, "Freedom 2.1 security issues and analysis," *White Paper, Zero Knowledge Systems, Inc*, 2001.

[71] O. Berthold, H. Federrath, and S. Köpsell, "Web mixes: A system for anonymous and unobservable internet access," in *Designing Privacy Enhancing Technologies*. Springer, 2001, pp. 115–129.

[72] D. Chaum, "The dining cryptographers problem: Unconditional sender and recipient untraceability," *Journal of cryptology*, vol. 1, no. 1, pp. 65–75, 1988.

[73] S. Goel, M. Robson, M. Polte, and E. Sirer, "Herbivore: A scalable and efficient protocol for anonymous communication," Cornell University, Tech. Rep., 2003.

[74] H. Corrigan-Gibbs and B. Ford, "Dissent: accountable anonymous group messaging," in *Proceedings of the 17th ACM conference on Computer and communications security*. ACM, 2010, pp. 340–350.

[75] M. K. Reiter and A. D. Rubin, "Crowds: Anonymity for web transactions," *ACM Transactions on Information and System Security (TISSEC)*, vol. 1, no. 1, pp. 66–92, 1998.

[76] M. K. Wright, M. Adler, B. N. Levine, and C. Shields, "The predecessor attack: An analysis of a threat to anonymous communications systems," *ACM Transactions on Information and System Security (TISSEC)*, vol. 7, no. 4, pp. 489–522, 2004.

[77] A. Beimel and S. Dolev, "Buses for anonymous message delivery," *Journal of Cryptology*, vol. 16, no. 1, pp. 25–39, 2003.

[78] M. J. Freedman and R. Morris, "Tarzan: A peer-to-peer anonymizing network layer," in *Proceedings of the 9th ACM conference on Computer and communications security*. ACM, 2002, pp. 193–206.

[79] G. Danezis and R. Clayton, "Route fingerprinting in anonymous communications," in *Peer-to-Peer Computing, 2006. P2P 2006. Sixth IEEE International Conference on*. IEEE, 2006, pp. 69–72.

[80] M. Rennhard and B. Plattner, "Introducing morphmix: peer-to-peer based anonymous internet usage with collusion detection," in *Proceedings of the 2002 ACM workshop on Privacy in the Electronic Society*. ACM, 2002, pp. 91–102.

[81] P. Tabriz and N. Borisov, "Breaking the collusion detection mechanism of morphmix," in *Privacy Enhancing Technologies*. Springer, 2006, pp. 368–383.

[82] D. Endres and J. Schindelin, "A new metric for probability distributions," *Information Theory, IEEE Transactions on*, vol. 49, no. 7, pp. 1858 – 1860, july 2003.

[83] J. Neyman and E. S. Pearson, "On the problem of the most efficient tests of statistical hypotheses," *Philosophical Transactions of the Royal Society of London Series A Containing Papers of a Mathematical or Physical Character*, vol. 231, no. 694-706, pp. 289–337, 1933.

[84] Y. Zhang and V. Paxson, "Detecting stepping stones," in *In Proceedings of the 9th USENIX Security Symposium*, 2000, pp. 171–184.

[85] D. L. Donoho, A. G. Flesia, U. Shankar, V. Paxson, J. Coit, and S. Staniford, "Multiscale stepping-stone detection: detecting pairs of jittered interactive streams by exploiting maximum tolerable delay," in *Proceedings of the 5th international conference on Recent advances in intrusion detection*, ser. RAID'02. Berlin, Heidelberg: Springer-Verlag, 2002, pp. 17–35.

[86] A. Blum, D. Song, and S. Venkataraman, "Detection of interactive stepping stones: Algorithms and confidence bounds," in *Recent Advances in Intrusion Detection*, ser. Lecture Notes in Computer Science. Springer Berlin / Heidelberg, 2004, vol. 3224, pp. 258–277.

[87] X. Wang and D. S. Reeves, "Robust correlation of encrypted attack traffic through stepping stones by manipulation of interpacket delays," in *Proceedings of the 10th ACM conference on Computer and communications security*, ser. CCS '03. New York, NY, USA: ACM, 2003, pp. 20–29.

[88] A. Houmansadr, N. Kiyavash, and N. Borisov, "RAINBOW: A robust and invisible Non-Blind watermark for network flows," in *Network and Distributed Systems Security Symposium*. Internet Society, Feb. 2009.

[89] A. Houmansadr and N. Borisov, "SWIRL: A scalable watermark to detect correlated network flows," in *NDSS*, 2011.

[90] Y. J. Pyun, Y. Park, D. S. Reeves, X. Wang, and P. Ning, "Interval-based flow watermarking for tracing interactive traffic," *Computer Networks*, vol. 56, no. 5, pp. 1646 – 1665, 2012.

[91] X. Wang, S. Chen, and S. Jajodia, "Network flow watermarking attack on low-latency anonymous communication systems," in *Security and Privacy, 2007. SP '07. IEEE Symposium on*, may 2007, pp. 116 –130.

[92] A. Serjantov and P. Sewell, "Passive-attack analysis for connection-based anonymity systems," *International Journal of Information Security*, vol. 4, pp. 172–180, 2005.

[93] B. Levine, M. Reiter, C. Wang, and M. Wright, "Timing attacks in low-latency mix systems," in *Financial Cryptography*, ser. Lecture Notes in Computer Science, A. Juels, Ed. Springer Berlin / Heidelberg, 2004, vol. 3110, pp. 251–265.

[94] L. Øverlier and P. Syverson, "Locating hidden servers," in *Security and Privacy, 2006 IEEE Symposium on*, may 2006, pp. 15 pp. –114.

[95] X. Wang, S. Chen, and S. Jajodia, "Tracking anonymous peer-to-peer voip calls on the internet," in *Proceedings of the 12th ACM conference on Computer and communications security*. ACM, 2005, pp. 81–91.

[96] W. Yu, X. Fu, S. Graham, D. Xuan, and W. Zhao, "Dsss-based flow marking technique for invisible traceback," in *Security and Privacy, 2007. SP '07. IEEE Symposium on*, may 2007, pp. 18 –32.

[97] S. J. Murdoch and G. Danezis, "Low-cost traffic analysis of tor," in *Proceedings of the 2005 IEEE Symposium on Security and Privacy*. Washington, DC, USA: IEEE Computer Society, 2005, pp. 183–195.

[98] Z. Ling, J. Luo, W. Yu, X. Fu, D. Xuan, and W. Jia, "A new cell-counting-based attack against tor," *Networking, IEEE/ACM Transactions on*, vol. 20, no. 4, pp. 1245–1261, 2012.

[99] P. Peng, P. Ning, and D. Reeves, "On the secrecy of timing-based active watermarking trace-back techniques," in *Security and Privacy, 2006 IEEE Symposium on*, may 2006, pp. 15 pp. –349.

[100] N. Kiyavash, A. Houmansadr, and N. Borisov, "Multi-flow attacks against network flow watermarking schemes," in *Proceedings of the 17th conference on Security symposium*, ser. SS'08. Berkeley, CA, USA: USENIX Association, 2008, pp. 307–320.

[101] X. Luo, P. Zhou, J. Zhang, R. Perdisci, W. Lee, and R. K. C. Chang, "Exposing invisible timing-based traffic watermarks with BACKLIT," in *Proceedings of the 27th Annual Computer Security Applications Conference*. ACM, 2011, pp. 197–206.

[102] Z. Lin and N. Hopper, "New attacks on timing-based network flow watermarks," in *USENIX Security Symposium*. Bellevue,WA: USENIX Association, Aug. 2012.

[103] C. Cachin, "An information-theoretic model for steganography," in *Information Hiding*. Springer, 1998, pp. 306–318.

[104] K. Loesing, W. Sandmann, C. Wilms, and G. Wirtz, "Performance measurements and statistics of Tor hidden services," in *The 2008 International Symposium on Applications and the Internet*. Turku, Finland: IEEE, July 2008, pp. 1 – 7.

[105] R. Fielding, J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach, and T. Berners-Lee, "Hypertext Transfer Protocol – HTTP/1.1," RFC 2616 (Draft Standard), Internet Engineering Task Force, Jun. 1999, updated by RFCs 2817, 5785. [Online]. Available: http://www.ietf.org/rfc/rfc2616.txt

[106] M. F. Arlitt and C. L. Williamson, "Web server workload characterization: the search for invariants," in *Proceedings of the 1996 ACM SIGMETRICS international conference on Measurement and modeling of computer systems*, ser. SIGMETRICS '96, 1996, pp. 126–137.

[107] N. Johnson, A. Kemp, and S. Kotz, *Univariate discrete distributions*, ser. Wiley series in probability and mathematical statistics. Applied probability and statistics. Wiley, 2005.

[108] L. J. Simon, "Fitting negative binomial distributions by the method of maximum likelihood," in *Proceedings of the Casualty Actuarial Society*, vol. XLVIII, Arlington, VI, 1961, pp. 45–54.

[109] S. Y. T. Soon, "Binomial approximation for dependent indicators," *Statistica Sinica*, vol. 6, pp. 703–714, 1996.

[110] S. J. Murdoch and P. Zieliski, "Sampled traffic analysis by internet-exchange-level adversaries," in *In Privacy Enhancing Technologies (PET), LNCS*. Springer, 2007.

[111] S. Haykin, *Neural Networks: A Comprehensive Foundation (2nd Edition)*, 2nd ed. Prentice Hall, Jul. 1998.

[112] P. C. Consul and F. Famoye, *Lagrangian probability distributions*. Birkhauser Boston, 2006.

[113] A. Biryukov, I. Pustogarov, and R.-P. Weinmann, "Content and popularity analysis of tor hidden services," *arXiv preprint arXiv:1308.6768*, 2013.

[114] Z. Botev, J. Grotowski, and D. Kroese, "Kernel density estimation via diffusion," *The Annals of Statistics*, vol. 38, no. 5, pp. 2916–2957, 2010.

[115] V. Paxson and S. Floyd, "Wide area traffic: the failure of poisson modeling," *IEEE/ACM Trans. Netw.*, vol. 3, no. 3, pp. 226–244, Jun. 1995.

[116] D. J. Olive, *Applied Robust Statistics*. Southern Illinois University, 2008. [Online]. Available: http://www.math.siu.edu/olive/ol-bookp.htm

[117] J. A. Elices and F. Pérez-González, "Measures to model delays on internet," http://www.unm.edu/~elices/captures.html, Jan. 2013.

[118] ISO, *ISO 3166-2:1998 Codes for the representation of names of countries and their subdivisions — Part 2: Country subdivision code*, 1998.

[119] L. Rizo-Dominguez, D. Munoz-Rodriguez, D. Torres-Roman, and C. Vargas-Rosales, "Packet variation delay distribution discrimination based on kullback-leibler divergence," in *Communications (LATINCOM), 2010 IEEE Latin-American Conference on*, sept. 2010, pp. 1 –4.

[120] L. Kleinrock, *Queueing Systems*. Wiley Interscience, 1975, vol. I: Theory.

[121] D. Kotz, T. Henderson, I. Abyzov, and J. Yeo, "CRAWDAD trace set dartmouth/campus/tcpdump (v. 2004-11-09)," http://crawdad.cs.dartmouth.edu/dartmouth/campus/tcpdump, Nov. 2004.

[122] Amazon Inc., "Amazon elastic compute cloud (amazon ec2)," http://aws.amazon.com/ec2/.

[123] T. Ylonen and C. Lonvick, "The Secure Shell (SSH) Protocol Architecture," RFC 4251 (Proposed Standard), Internet Engineering Task Force, Jan. 2006. [Online]. Available: http://www.ietf.org/rfc/rfc4251.txt

[124] P. Moulin and J. O'Sullivan, "Information-theoretic analysis of information hiding," *Information Theory, IEEE Transactions on*, vol. 49, no. 3, pp. 563–593, 2003.

[125] M. Barni and B. Tondi, "The source identification game: An information-theoretic perspective," *IEEE Transactions on Information Forensics and Security*, vol. 8, no. 3, pp. 450–463, 2013.

[126] T. Basar and G. J. Olsder, *Dynamic Noncooperative Game Theory (Classics in Applied Mathematics)*, 2nd ed. Soc for Industrial & Applied Math, Jan. 1999.

[127] G. Casella and R. Berger, *Statistical inference*, ser. Duxbury advanced series in statistics and decision sciences. Thomson Learning, 2002.

Appendices

Appendix A

Theoretical Probabilities for IPD-based without traffic modification

A.1 Cauchy Test

Recall from (6.6) that we decide the flows are correlated if the random variable W defined as

$$W = \prod_{i=1}^{n-1} \frac{\overbrace{(\Delta Y_i)^{\alpha+1}}^{V_i}}{\pi\sigma\alpha x_m^{\alpha} \underbrace{\left(1 + \left(\frac{\Delta Y_i - \Delta X_i}{\sigma}\right)^2\right)}_{U_i}} \tag{A.1}$$

is larger than η. Also notice that in (A.1) we have defined the auxiliary random variables U_i and V_i.

As shown in Section 6.3, when the flows are uncorrelated the ΔX_i and ΔY_i become independent Pareto-distributed random variables. Nevertheless, under the Cauchy model assumption, $\Delta Y_i - \Delta X_i$ follows a Cauchy distribution when flows are

correlated. Therefore the distribution of U_i can be obtained through a transformation of random variables (see [127]) as follows:

$$f_{U|H_0}(u) = \int_{x_m}^{\infty} \frac{\sigma}{2\sqrt{u-1}} \frac{\alpha x_m^{\alpha}}{z^{\alpha+1}} \left(\frac{\alpha x_m^{\alpha}}{(z+\sigma\sqrt{u-1})^{\alpha+1}} + \frac{\alpha x_m^{\alpha}}{(z-\sigma\sqrt{u-1})^{\alpha+1}} \right) dz \tag{A.2}$$

and

$$f_{U|H_1}(u) = \frac{1}{\pi u\sqrt{u-1}}. \tag{A.3}$$

V can be characterized as follows:

$$f_V(v) = \frac{1}{v^2\pi\sigma} \int_{x_m}^{\infty} f_U \left(\frac{z^{\alpha+1}}{v\sigma\pi\alpha x_m^{\alpha}} \right) dz. \tag{A.4}$$

Lastly, we characterize W, as $f(w) = f_{v_1 \cdot v_2 \cdots v_{n-1}}(w)$, where the density of the product of two independent random variables can be obtained as

$$f_{V_1 \cdot V_2}(v) = \int_{-\infty}^{\infty} \frac{1}{z} f_{V_1}(\frac{v}{z}) f_{V_2}(z) dz. \tag{A.5}$$

So we calculate the value of the threshold, η, as the $(1 - P_F)$th quantile of $f_{W|H_0}$ and the probability of detection as $P_D = 1 - F_{W|H_1}(\eta)$, where F_W denotes the cumulative distribution function of f_W.

A.2 Laplace Test

Under Laplace assumption, the test (6.7) is

$$W = \prod_{i=1}^{n-1} \underbrace{\overbrace{\exp\left(-\frac{|\Delta Y_i - \Delta X_i|}{\sigma} \right)}^{U_i} \frac{(\Delta Y_i)^{\alpha+1}}{2\sigma\alpha x_m^{\alpha}}}_{V_i} \tag{A.6}$$

in the case that W is larger than η we decided the flows are correlate. Similarly to the Cauchy case, we have defined the auxiliary random variables U_i and V_i.

The only difference between the Cauchy model of the previous section is that $\Delta Y_i - \Delta X_i$ follows a Laplace distribution when flows are correlated. Using a transformation of random variables the distribution of U_i becomes

$$f_{U|H_0}(u) = \int_{x_m}^{\infty} \frac{\alpha^2 x_m^{2\alpha}}{z^{\alpha+1}} \left(\frac{1}{(v - \sigma \log u)^{\alpha+1}} + \frac{1}{(v + \sigma \log u)^{\alpha+1}} \right) dz \tag{A.7}$$

and $U|H_1$ becomes a uniform random variable in the interval $[0, 1]$.

V can be characterized as follows:

$$f_V(v) = 2\sigma\alpha^2 x_m^{2\alpha} \int_{x_m}^{\infty} f_U \left(\frac{2v\sigma\alpha x_m^{\alpha}}{z^{\alpha+1}} \right) \frac{1}{z^{2(\alpha+1)}} dz. \tag{A.8}$$

As in previous section $f(w) = f_{v_1 \cdot v_2 \cdots v_{n-1}}(w)$, where the density of the product of two independent random variables can be obtained using (A.5). The threshold, η, is $(1 - P_F)$th quantile of $f_{W|H_0}$ and $P_D = 1 - F_{W|H_1}(\eta)$, where F_W denotes the cumulative distribution function of f_W.