Beyond TOR: The TrueNyms Protocol

Nicolas Bernard and Franck Leprévost

University of Luxembourg, LACS, 162 A, Avenue de la Faïencerie,
L-1511 Luxembourg
{Nicolas.Bernard,Franck.Leprevost}@uni.lu

Abstract. How to hide who is communicating with whom? How to hide when a person is communicating? How to even hide the existence of ongoing communications? Partial answers to these questions have already been proposed, usually as byproducts of anonymity providing systems. The most advanced one available today is Onion-Routing and is implemented in Tor and I2P. Still, Onion-Routing is exposed to a series of serious attacks. The current paper classifies these series of attacks, and announces the TrueNyms unobservability protocol. We describe here how TrueNyms handles one of the families of attacks applying to the current Onion-Routing system, namely traffic analysis on the "shape", and give some evidence on its performance. Developed since 2003, TrueNyms is not anymore an academic answer to a privacy problem, but is a heavily tested and efficient product providing unobservability and anonymity. Although it cannot be used (for the time-being) for *very low-latency* applications like telephony over IP, TrueNyms can be efficiently used for most *low-latency* applications like Web browsing and HTTP-based protocols (RSS for instance), Instant Messaging, File transfers, audio and video streaming, remote shell, etc. TrueNyms allows parties to communicate without revealing anything about the communication — including its very existence — to any observer, despite how powerful such an observer might be.

1 Introduction

For low-latency communications, protocols like SSL [25] or IPsec [16] allow the encryption of data and authentication of parties. Still, these protocols do not protect all aspects of communication over the Internet. Notably, an observer is still able to learn the identity of the communicating parties, or the nature of the content (Web browsing, file transfer, VoIP, etc.).

The questions we address here are: How to hide who is communicating with whom? How to hide when a person is communicating? How to even hide the existence of ongoing communications?

Partial answers to these questions have already been proposed, usually as byproducts of anonymity providing systems. The most advanced one available today is Onion-Routing [6,12,24] and is implemented in Tor [8] and I2P [15]; enhancements to these systems have been proposed (e.g. [27]). Still, Onion-Routing is not sufficient, and is exposed to a series of serious attacks.

P. Bouvry et al. (Eds.): SIIS 2011, LNCS 7053, pp. 68–84, 2012.

In this paper, we recall in section 2 how Onion-Routing works, raise the performance issues, and classify its security issues into three families of attacks. In section 3 we address one of these security issues, namely preventing traffic analysis on the "shape", and introduce our protocol TrueNyms. Although based on Onion-Routing, TrueNyms bypasses all its security drawbacks [3]. In section 4 we provide the first performance measurements of our protocol after a very intensive testing phase. As a consequence, TrueNyms provides a concrete, practical, and efficient answer to the questions addressed above, which can be resumed as looking for the missing link to privacy. In other words:

- TrueNyms solves all the security issues applying to Onion-Routing;
- In terms of perfomance, although TrueNyms's latency suffers from the security improvements over Onion-Routing, TrueNyms is more efficient than the Onion-Routing implementations (Tor, and I2P), as far as the establishment of communications, and the throughput are concerned ;
- The current version of TrueNyms can be used for most applications like Web browsing and HTTP-based protocols (RSS for instance), Instant Messaging, File transfers, audio and video streaming, remote shell, etc. The applications excluded (for the time-being) are those needing a very low latency (like telephony over the Internet).

Henceforth, with TrueNyms, it is now possible to communicate without revealing anything about the communication — including its very existence — to any observer (passive or active), as powerful as such an observer may be.

A more complete description of our TrueNyms protocol will appear elsewhere [3], where we will further detail the adopted methods not only against traffic analysis but against all three families of attacks applying to Onion-Routing, the in-depth security architecture of the TrueNyms program itself, and further performance data.

2 Description of Onion-Routing and Its Weaknesses

Alice and Bob want to communicate in a very secure way. They want to keep secret not only the content of their communications, but the very fact that they are communicating should be itself a secret too: they want *unobservability*. We suppose nothing about the observers, which may be the computer engineer in the company you are working in, the intelligence service of a very powerful nation-state, or even a combination of such intelligence services like Echelon (see e.g. [9], especially Chapter 2, and [5]). The monitoring of the observers can be targeted on Alice and / or Bob or can be a global and ubiquitous system. Eve is a passive observer; Mallory is an active one.

2.1 From Encryption to Onion-Routing

Let us recall that using some encryption to ensure unobservability is not sufficient. It usually protects only the content of the data packets, and not the

headers. As a consequence, an observer sees that the communication takes place between Alice and Bob.

A natural approach to overcome this weakness is to introduce a *relay* R between Alice and Bob. The sender Alice first encrypts her original message M with Bob's key k_B, then encrypts the result $\{M\}_{k_B}$ with the relay's key k_R, and finally sends the newly encrypted message $\{\{M\}_{k_B}\}_{k_R}$ to the relay. Now, when the relay R receives this message, he removes one layer of encryption, recovers hence the message $\{M\}_{k_B}$, which he sends to Bob, who in turn is able to recover the original message M sent by Alice. With this approach, the message sent by Alice to R is distinct from the message resent by R to Bob, and hence an observer cannot *a priori* be able to make the link between the messages sent to R, and the messages sent by R. Moreover, the headers containing the IP addresses of the sender and of the receiver, added at the beginning of each encrypted packet, are *a priori* misleading the observer. Such systems are available on a commercial basis, but with encryption only between Alice and the relay R (see e.g. `anonymizer.com`, `swissvpn.net`, `xerobank.com`).

However, this approach suffers from some serious drawbacks, as R knows the identities of both Alice and Bob, who furthermore must trust R.

2.2 Brief Description of Onion-Routing

One can rely on nested tunnels established through multiple relays R_1, R_2, etc. (in what follows, a *node* denotes either a relay or Alice or Bob). These relays accept to take part in an anonymity system, but are not necessarily trusted. Indeed, some of them can cooperate with Eve or Mallory. Relays see only enciphered traffic and know only the previous and next nodes on the route. They do not know if those nodes are other relays or end-points. This approach, known as Onion-routing (see [12,24]), or as Pipenet [6], and is illustrated in Figure 1.

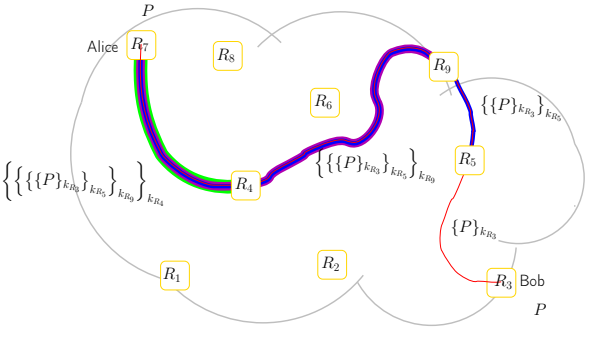

Fig. 1. illustrates how Onion-Routing works, using the notations of 2.1 in an obvious way: to communicate with Bob, Alices creates a set of nested encrypted tunnels. For every packet, each relay removes the outermost encryption layer (hence the name of this scheme).

Throughout this article, an encrypted tunnel between Alice and one of the nodes is called a *connection*. Then, a set of nested connections between Alice and Bob is called a *route*. Despite being created by Alice, those routes are not related to IP source routing or other IP-level routing. Standard IP routing is still used between successive nodes if these nodes are on an IP network as we consider here. A *communication* is a superset of one or more routes between Alice and Bob that are used to transmit data between them. A communication can use multiple routes simultaneously and / or sequentially.

The concept of Onion-Routing goes back to 1996, and is the most advanced system available today. A few implementations of Onion-Routing exist, the better known being probably Tor [8] and I2P [15]. Still, Onion-Routing is not sufficient to achieve unobservability. Onion-routing is subject to a set of serious attacks, and has also performance issues described in the following sections.

2.3 Performance Issues

Establishment time, latency and throughput lead to performance issues for Onion-Routing.

The **establishment time** is the first issue in Onion-Routing. If Alice creates a route with n relays to communicate with Bob, she has to create $n+1$ connections instead of only one as she would do in a classical TCP-connection. Moreover, the time to establish these connections is impacted by key-establishment and partial authentication protocols.

The **latency** considered here is the time for a packet to go from one end of the route to the other end. Latency depends mostly on the path taken by the packets on the network. With a standard communication using standard routing mechanisms, latency between Alice and Bob would probably be low if Alice and Bob are close (in a network sense) to each other. However, with Onion-Routing, the total latency L_{AB} between Alice and Bob is the sum of the latencies between each couple of consecutive nodes on the route (notwithstanding some processing time), and is given by the following formula, assuming there is an average latency $\hat{L}$ between two nodes:

$$L_{AB} = L_{AR_1} + \sum_{i=1}^{n-1} L_{R_i R_{i+1}} + L_{R_n B} \approx (n+1)\hat{L}\,. \tag{1}$$

Finally, the **throughput** between two nodes measures the amount of data transmitted between these nodes per time unit. Clearly, the throughput T_{AB} between Alice and Bob is limited by the lowest throughput between two successive nodes:

$$T_{AB} \leq \min\left\{T_{AR_1}, \left\{T_{R_i R_{i+1}}, 1 \leq i < n\right\}, T_{R_n B}\right\}\ . \tag{2}$$

These three issues have multiple consequences, ranging from minor annoyances to the user (delays, slower downloads, etc.) to the point where some protocols can hardly be used (e.g. Telephony over IP) with Onion-Routing.

2.4 Classification of Security Issues

The security issues of Onion-Routing may be classified in three categories of attacks:

- Analysis of connection creation;
- Replay attacks;
- Analysis of the shape of the traffic.

None of these attacks is exactly new [1,7,11,22,26,28,29,30,33,34], but no low latency system was solving all of them before TrueNyms.

While it would be easier to perform these attacks by monitoring the whole network, note that it is not needed to make pretty accurate guesses. Even if the observer (passive or active) is not able to observe every node or even every node on a route, correlation on the connections going through the part(s) of the network the observer monitors will give him important information, notably if both Alice and Bob are part of the area the observer looks at [21].

Beginning / End of Connections. Let us assume that Eve can view a part of the network, and let us consider Figure 2, representing this part. When a

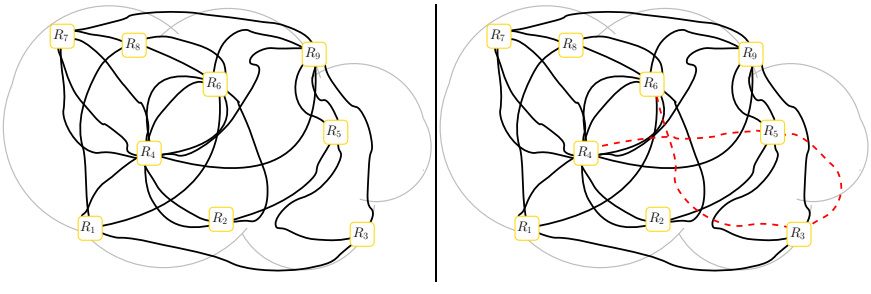

Fig. 2. Background activity at time t (left) and a new route (right)

route is established over the network, even if there are other routes, there will be little change in these while the new route is established if the connections are made without pausing. For instance, let us assume that at some point the network looks like the left part of Figure 2. Then imagine that an activity is ongoing leading to the situation depicted on the right, with the establishment of a connection between the node R_4 and the node R_5, then from R_5 to R_3, then R_3 to R_6, and assume that the background picture of the network stays globally invariant like in the left part of Figure 2. If there is no further activity for some times, Eve will guess that the dashed route has just been established between the nodes R_4, and R_6. The situation is similar when a route is no longer needed and is destroyed.

Replay Attacks. An issue with standard cryptography modes when used in Onion-Routing is that they allow an active replay attack[1]. Let us examine the situation at a relay at a given time: for instance, let us assume that this specific relay is a part of three routes, as depicted in Figure 3.

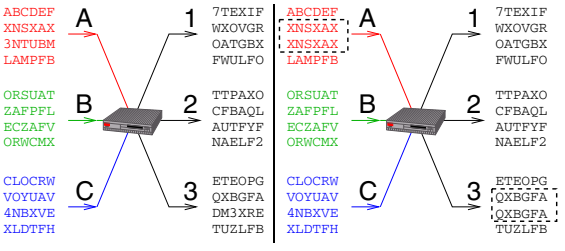

Fig. 3. Cryptography hides connection bindings to a passive observer (left), but not to an active observer able to inject duplicate packets (right)

On the left of Figure 3, Mallory sees three distinct incoming connections (A, B, C). As an encryption layer is removed on each connection, he does not know the corresponding outgoing connections. However, as cryptography is deterministic, a given packet entered twice through the same incoming connection will be output twice — in its form with an encryption layer removed — on the corresponding outgoing connection. So Mallory takes a packet and duplicates it, say on the connection A, which leads to the right side of Figure 3. He then looks for two identical packets on the output, and finds them on the connection 3, so he learns that connection A and connection 3 are part of the same route. Obviously, depending on the interest of Mallory, he can perform a similar attack on the next relay having the connection 3 as an incoming connection, and then see where this leads ultimately. Or he can perform the same attack on the other incoming connections B and C, and figure out exactly which outgoing connection 1 or 2 corresponds to them. Other attacks, depending on the encryption mode, are also considered in [3].

Analysis of the "Shape" of the Traffic. The third class of security issues stems from what could be called "the shape" of the traffic (see e.g. [26]). Indeed, each connection has a very specific signature given by a graph having as x-coordinate the moment when a packet is observed, and its size as y-coordinate. Moreover, this signature shape reveals, despite the encryption, the kind of traffic passing over this connection. For instance, figure 4 illustrates this, where the incoming connection A (resp. connection B, resp. connection C) probably encapsulates Web traffic (resp. VoIP, resp. File transfer).

[1] This is different of the replay attacks well known in cryptography, where an attacker can play part of a protocol back from a recording, and that are usually prevented by the use of nonces or timestamps.

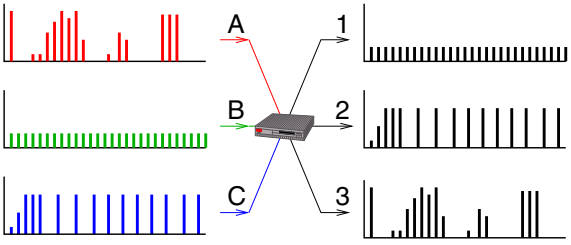

Fig. 4. It is easy to match connections when there is no traffic shaping

Obviously, Eve can easily match the shapes of the incoming connections to the shapes of the outgoing ones to discover that connection A corresponds to connection 3, connection B to connection 1, and connection C to connection 2.

In fact, if an attacker is only able to observe Alice (i.e. far less powerful than what we have assumed so far) and hence is not able to observe any relay, he still can use the shape of the traffic to know what Alice does, for instance by combining attacks like those described in [4,13,14,17,31,32].

3 The TrueNyms Protocol and Its Approach to Traffic Analysis

The previous section described three families of security issues in Onion-Routing that must be solved. This section considers only the last one (a more complete description of the countermeasures against the other families of attacks will appear in [3]), and introduces our TrueNyms protocol.

The shape of the traffic on a connection depends on the specific number of packets this connection sends and receives, the size of each packet, and the time distribution of the packets being related to the ongoing traffic. While it is easy to impose a specific shape of the traffic, this has consequences that have to be addressed, as we see below.

3.1 Traffic Shaping

The problem of the different size for the packets can be solved by imposing the size: each and every packet will be of the same size. Shorter packets will be padded before encryption, bigger packets will be split in two or more. Similarly, it is possible to impose a specific number of packets per second on every connection. Doing these two operations is *traffic shaping*.

However, while it solves the visibility of the shape problem, traffic shaping has its own issues that must, in turn, be solved. The first one is an optimization problem: if both packet size and the number of packets per second are imposed, what are the optimal parameters for these? Although the answer can be tricky (see [3]), let us however assume that we have fixed reasonable parameters. Still, two security issues remain:

– first, while the input-output link is not immediate anymore, the issue of the total number of packets on connections is not solved yet. The way this problem is addressed in TrueNyms is described in [3];
– second, the loss, or even the delay, of a single packet can expose the hidden link. This issue and how TrueNyms solves it is described in 3.2.

3.2 What Happens When a Packet Is Lost?

When a packet is lost (and some will be as experience shows), if nothing is done to prevent it, the situation depicted in Figure 5 arises. The relay has nothing to

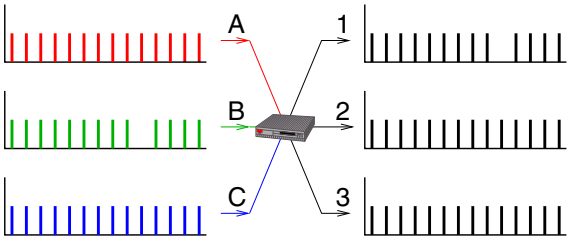

Fig. 5. Lost packets are an issue with traffic shapping

send when it should send the missing packet, and hence sends nothing. Observing this, Eve immediately knows connection B and 1 are parts of the same route. She may even be able to "follow" the missing packet from relay to relay until it "reaches" the end of the route.

A naive approach to solve this issue could be for the relay to drop a packet on each outgoing connection. While it would prevent an observer to see the link, it would amplify the loss of a single packet to all the routes going through this relay, and would lead to obvious problems.

A more suitable approach is for the relay to insert a dummy packet (i.e. consisting of random data) each time a packet is lost. Its content being random, this packet cannot be distinguished from a normal, enciphered packet, by anyone but its destination, either Alice or Bob.

3.3 Issues with Dummy Packets

Assumptions. Let us now clarify the assumptions used, and their first consequences. The idea is to build an *application* allowing unobservable communications. It needs to be an application because the assumptions include:

1. it is not possible to change the core network infrastructure (routers, etc.);
2. users will not change their operating system only to have unobservability.

These assumptions imply the system cannot be at the operating system kernel level (or else it would imply a lot more development work). The system cannot be a mere library either: the need to hide its global use has to be coordinated between applications in the case more than one of them are running, and has also to be ensured in the case no application is running.

Implicit Model. Being an application, the logical choice is to use between relays the default communication mechanism, namely TCP sockets, *i.e.* standard reliable (but insecure) stream oriented connections: this is the implicit model. The reliability of TCP means (apart network failure), all data put at one end of the connection arrive at the other end, the lost packets being retransmitted. If we represent two successive relays on a route, the situation is as depicted in Figure 6.

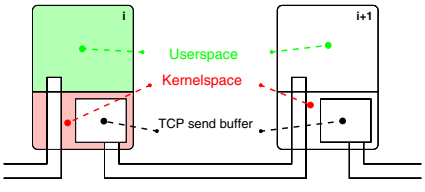

Fig. 6. Implicit model up to now: application level forwarding program linked through TCP connections

TCP is handled at the operating system kernel level ("kernelspace"), while the relaying process is an application (in the "userspace"). The operating system of a relay stores a sent packet in a "send buffer" until the next relay acknowledges its reception (then, it is deleted). While TCP is reliable, the need for reliability on a route from one end to the other end implies that the relaying application must be reliable too, *i.e.* it must not lose or discard packets.

The implicit model will be *discarded* as we need to send dummy packets. Indeed, let us consider what happens with the implicit model if a dummy packet is inserted when a packet is "lost" (or delayed, as reliable underlying TCP connections are used). On each relay, the program alternates between a reading phase, where it receives packets on the network, and a writing phase, where, for each connection, *one and only one* packet is sent. If multiple packets are received on a connection, one only is sent, the other ones are queued. If no packet is received, a dummy one is created and sent.

Let us assume at time t (see Figure 7), that both programs are in their reading phases and receive a packet on each of the two pictured relays (relays i and $i+1$)[2].

[2] Things are presented as synchronized to make the picture clearer. However there is no real synchronization in the actual system, but for a common parameter for the number of packets per second. Clock skew is not an issue, neither at the timescale of a connection (too small) nor between connections (irrelevant).

At time $t+1$, both programs send the previously received packet. Unbeknownst to relay i, the packet it sent is lost. At time $t+2$ (Figure 8, left), both programs

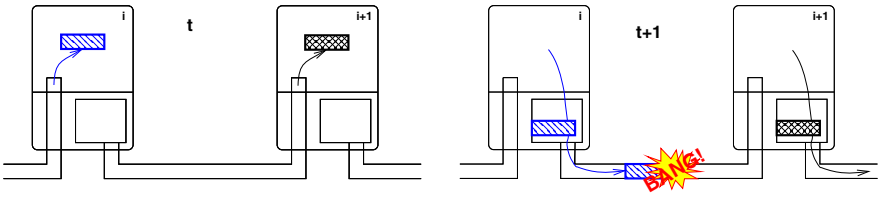

Fig. 7. Packets received (left) are then forwarded (right), but one of them is lost

are, again, in their reading phase. The program on node i gets a packet, but the program on node $i+1$ gets none, as the packet was lost. Meanwhile, the operating system of node $i+1$ receives an "ACK" from node $i+2$, and so it discards the copy of the packet it still had in its send buffer: at time $t+3$ (Figure 8, right),

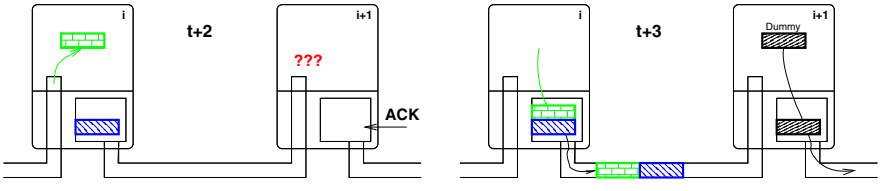

Fig. 8. Due to the lost packet, node $i+1$ does not receive data in time (left). It must then send a dummy packet (right).

both programs are in a new writing phase. Relay program i just sends the packet it got at time $t+2$, which means it gives the packet to the operating system, requesting its delivery to relay $i+1$. The operating system still has the copy of the previous packet in its send buffer. The exact timing can vary, but in essence, the operating system will then send both the new and the old packet on the network. While there are two packets on a connection at the same time, this is not by itself a security issue, as long as there is no correlation with what appeared, appears, or will appear on other connections. Meanwhile, node $i+1$ has to send a packet, but it has received none in the imparted time. Hence, it creates a dummy packet with random data, and sends it. At time $t+4$ (Figure 9, left), both programs are again reading. Relay i receives a new packet, and relay $i+1$ gets at the same time the two packets sent by i. Here is the issue: at time $t+5$ (Figure 9, right), each program, must send one and *only one* packet. This is no issue for relay i, but relay $i+1$ has to send one packet and to queue

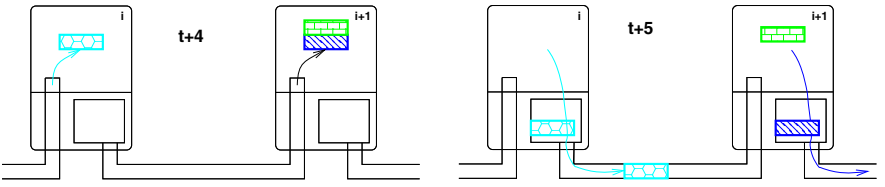

Fig. 9. Relay $i+1$ now receives two packets at the same time (left). But it can send only one (right).

the other one. In the following instants, it will receive the packet sent by relay i. Then it will be able to send the packet in its queue, but then will have to add the new packet, etc. Relay $i+1$ will have to juggle with packets hereafter.

The issue with the implicit model is twofold:

- first, the FIFO queues on the nodes have an impact on the latency: each lost packet, even if it was a dummy packet, increases the latency on the route overtime;
- second, it uses memory; one can imagine a denial of service attack where Mallory ensures some specific packets are lost so that some router is out of memory.

The solution is to abandon the implicit model, and adopt an unreliable model which allows relays to discard packets (while still insuring end-to-end reliability between Alice and Bob).

3.4 Handling Reliability: Our Unreliable Model

The problem caused by sending a dummy packet over a reliable connection as described in 3.3 has no obvious solution. When designing TrueNyms, the choice was made to change the model and to build our system over unreliable channels, leading to our unreliable model. Once relays are able to discard packets and do not have a FIFO queue, the issue caused by dummy packets disappears. It is not even mandatory to change the underlying communication protocol away from TCP. However, except for some cases where it should be used and not another protocol, TCP is now only a second choice. Indeed, the reliability TCP offers between two relays is useless. What counts is the connection of each node of the route with Alice and, except on the first relay, that packets can have been discarded by a previous relay on this connection.

Worse, when a packet is lost between two relays, TCP will retransmit it. However, while the packet is retransmitted, the relay program will not be able to get anything: the packets following the one that was lost are delayed until it is retransmitted. During this waiting time, the relay has to generate dummy packets for traffic shaping. When at last the missing packet is retransmitted, it is given

to the program, *at the same time as all the subsequent packets arrived meanwhile*. As the program is anyway unable to forward them all, it will discard most of them. The loss of a packet that may have been a dummy packet is so transformed in the loss of multiple packets, including possibly non-dummy packets. So: *in an unreliable model, TCP must be considered harmful*. In TrueNyms, TCP is only used when another, unreliable, protocol cannot be used.

While either or both the underlying protocol and the relays are able to lose packets, end-to-end reliability is still needed in most cases. To the initial implicit model of host-to-host reliability, TrueNyms substitutes an unreliable model that still provides end-to-end reliability between Alice and Bob. However, even this end-to-end reliability is optional in TrueNyms. This means that when Alice and Bob are communicating through a protocol that does not need end-to-end reliability, it is possible to discard this end-to-end reliability, to the profit of latency for instance. Being able to provide both reliable and unreliable transport to the upper layers, TrueNyms' conceptual position in the TCP/IP network model can be located between IP and the applicative layer. However, to have a better compatibility with existing networks and networking applications (packet filters, etc.), TrueNyms is actually built upon UDP (with TCP as a backup).

This section described how TrueNyms solves the issue of the shape of the traffic. Actually TrueNyms solves the other families of problems as well [3] and is a complete system.

4 Implementing TrueNyms: Addressing Performance Issues

Implementation of TrueNyms started in the last quarter of 2003 as a prototype. TrueNyms is implemented in the C language (ISO C99) and runs on Unix / Posix systems. TrueNyms consists in a peer-to-peer daemon, to which native clients can connect. A library is provided to write native clients. It is, however, not necessary for most uses: a SocksV5 proxy is provided that allows existing software to interface with the daemon.

Regarding the choice of the parameters for our tests, we chose *a priori* sensible values for a generic use of TrueNyms, with 10 packets per seconds and a packet size of 1044 bytes (when over UDP). This packet size gives MTUs of 988 and 996 bytes for respectively reliable and unreliable packets in our unreliable model. Latency could be decreased by sending more packets per second. Unless the size of each packet is decreased, it would also increase the throughput per route, but would increase the burden on the network and disallow the use of the TrueNyms system on slower accesses (like dial-up POTS connections for instance).

During the whole development, the program was constantly tested on a LAN. To gain insights on performance over a real network, we also later deployed (safely) the program on distinct locations linked through the Internet.

4.1 Test Network

Such a test deployment was done by sending small (and secure) computers to miscellaneous universities through Europe (University of Luxembourg, Technische Universität Berlin, University Joseph Fourier in Grenoble, École Normale Supérieure in Lyon, University of Liège, University of Namur). All these universities are connected to the Internet through their national research network, which in turn are themselves connected together through the European research network GÉANT 2. The network links between the relays were both low latency, with an average round-trip time $\hat{t}_{rtt}$ between two relays of about 30 ms, and high throughput. We tested and optimized TrueNyms over this network during more than two years.

4.2 Performance

We measured the performances of the latency and of the time needed to establish a communication.

Latency. With standard Onion-Routing, latency L_{OR} can be approximated by

$$L_{OR} = n \times \hat{t}_{rtt} + 2n \times t_{proc}\,, \tag{3}$$

where n is the number of relays, $\hat{t}_{rtt}$ the average round-trip time between two relays and t_{proc} the average processing time of a packet on a relay. However, when traffic shaping is used, the new important parameter $\hat{t}_{wait}$ must be considered. On average, if the duration between two send phases is called t_{sl}, there is a waiting time of $\hat{t}_{wait} = \frac{t_{sl}}{2}$. As a consequence, TrueNyms' latency L_{TN} becomes

$$L_{TN} = n \times \hat{t}_{rtt} + 2n \times t_{proc} + 2n \times \hat{t}_{wait}\,. \tag{4}$$

Obviously, the dominant term depends on both the chosen parameters, the network and the computers used as relays. On the testbed network, and with a configuration specifying to send 10 packets/sec, the parameters were the following:

$$\hat{t}_{rtt} \approx 30 \text{ ms}, \qquad t_{proc} \approx 5 \text{ ms}, \qquad \hat{t}_{wait} \approx 50 \text{ ms}\,. \tag{5}$$

The graph in Figure 10 shows expected latency as well as latency measured over our tests of TrueNyms. This shows that while this kind of latency will not allow to use some protocols needing a very low latency (e.g.Telephony), TrueNyms is still suitable for most uses with three relays on a route (which provides a more than adequate security).

Time to Establish a Communication. We consider this aspect as important as latency. Indeed, even if both throughput and latency were excellent, if it required to wait a few minutes between the time a request is sent and the time the answer arrives, it would restrict the use of the system to a few aspects (e.g. automated transfers). However, as described in [3], TrueNyms uses a pool of pre-established

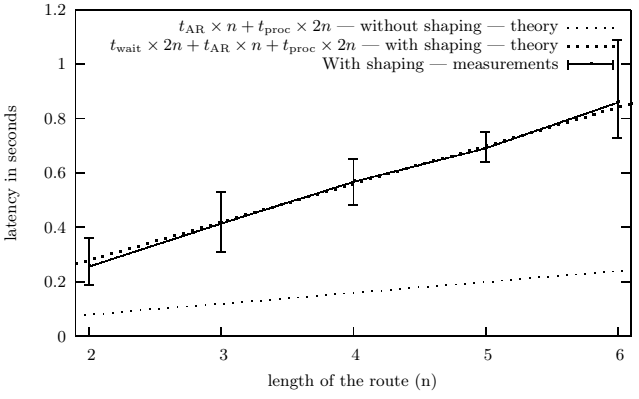

Fig. 10. Theoretical and measured latency for TrueNyms

proto-routes to accelerate the establishment of communications. With six relays, about 90 seconds are needed on average to establish a route. The pool of routes provides a dramatic improvement: it reduces this establishment time to around one second (average 0.75 seconds with three relays, 1.13 with six).

5 Conclusions

The TrueNyms protocol allows Alice and Bob to communicate without any observer knowing it. When parties are using TrueNyms for their communications, an observer, as powerful as he may be, is unable to know who they are communicating with. He is unable to know when a communication occurs. He is even unable to know if a communication occurs at all.

TrueNyms is based on Onion-Routing, to which it adds protection against all forms of traffic analysis. With the current knowledge, it seems out of reach to achieve a formal proof of security of such a system. However, some tried and true principles exist to minimize the risks. These principles and concepts were applied during the design of TrueNyms to make it future-proof, and harden it against yet unknown attacks. To date, this approach was fruitful: since we began our work on TrueNyms, other attacks have been discovered against Onion-Routing or Tor. Some are specific to Tor, due to specific features integrated in it but not related to Onion-Routing [18,19,23]. Others are related to Onion-Routing but, again, specific to Tor. They are due, for instance, to optimizations made to enhance Tor's performance, which in fact lead to the introduction of flaws [2]. At last, some are pretty generic, like [20,10]. While we did not foresee these attacks, none of them worked against TrueNyms, without changing a single line of the code. This is due to the "security over performance" and "security in depth" approaches used when designing TrueNyms. TrueNyms was extensively tested for more than

two years on a test network. Its performance is experimentally validated and is appropriate for most uses : Web browsing and HTTP-based protocols (RSS for instance), Instant Messaging, File transfers, audio and video streaming, remote shell, etc.

While focused on *unobservability*, for situations where communicating parties may want to authenticate themselves at a upper layer, TrueNyms provides *anonymity* as a byproduct. This anonymity is at the network level, and is much stronger than what other comparable systems (like e.g. Onion-Routing) provide. The paper [3] completes the present article in providing much more details on TrueNyms, its implementation, its in-depth security, and its performances.

Acknowledgements. The authors thank P. Bouvry, M. Muraszkiewicz, F. Seredynski for their careful reading of [3], and the referees, as well as the Technische Universität Berlin, the University Joseph Fourier of Grenoble, the École Normale Supérieure of Lyon, the University of Liège and the University of Namur for joining the first test campaign. The FNR/04/01/05/TeSeGrAd grant partially supported this research.

References

1. Back, A., Möller, U., Stiglic, A.: Traffic analysis attacks and trade-offs in anonymity providing systems. In: Moskowitz, I.S. (ed.) IH 2001. LNCS, vol. 2137, pp. 245–257. Springer, Heidelberg (2001)
2. Bauer, K., McCoy, D., Grunwald, D., Kohno, T., Sicker, D.: Low-resource routing attacks against Tor. In: Proceedings of the Workshop on Privacy in the Electronic Society (WPES 2007), Washington, DC, USA (October 2007)
3. Bernard, N., Leprévost, F.: Unobservability of low-latency communications: the TrueNyms protocol. Work in Progress (2011)
4. Bissias, G.D., Liberatore, M., Jensen, D., Levine, B.N.: Privacy vulnerabilities in encrypted HTTP streams. In: Danezis, G., Martin, D. (eds.) PET 2005. LNCS, vol. 3856, pp. 1–11. Springer, Heidelberg (2006)
5. Campbell, D.: A new way to do anonymity. STOA European Parliament 168.184/Part.4 (April 04, 1999)
6. Dai, W.: A new way to do anonymity. Post to Cypherpunks Mailing List (February 07, 1995)
7. Danezis, G.: The traffic analysis of continuous-time mixes. In: Martin, D., Serjantov, A. (eds.) PET 2004. LNCS, vol. 3424, pp. 35–50. Springer, Heidelberg (2005)
8. Dingledine, R., Mathewson, N., Syverson, P.: Tor: The second-generation onion router. In: Proceedings of the 13th USENIX Security Symposium (August 2004)
9. Ebrahimi, T., Leprévost, F., Warusfel, B. (eds.): Enjeux de la sécurité multimédia. Informatique et Systèmes d'Information, Hermes-Lavoisier (2006)
10. Evans, N., Dingledine, R., Grothoff, C.: A practical congestion attack on tor using long paths. In: Proceedings of the 18th USENIX Security Symposium (August 2009)
11. Fu, X., Graham, B., Bettati, R., Zhao, W.: Active traffic analysis attacks and countermeasures. In: Proceedings of the 2003 International Conference on Computer Networks and Mobile Computing, pp. 31–39 (2003)

12. Goldschlag, D.M., Reed, M.G., Syverson, P.F.: Hiding Routing Information. In: Anderson, R. (ed.) IH 1996. LNCS, vol. 1174, pp. 137–150. Springer, Heidelberg (1996)
13. Herrmann, D., Wendolsky, R., Federrath, H.: Website fingerprinting: attacking popular privacy enhancing technologies with the multinomial naïve-bayes classifier. In: Proceedings of the 2009 ACM Workshop on Cloud Computing Security (CCSW 2009), pp. 31–42. ACM, New York (2009)
14. Hintz, A.: Fingerprinting websites using traffic analysis. In: Dingledine, R., Syverson, P.F. (eds.) PET 2002. LNCS, vol. 2482, pp. 171–178. Springer, Heidelberg (2003)
15. The Invisible Internet Project: Introducing I2P (200x), `http://www.i2p2.de/`
16. Kent, S., Atkinson, R.: RFC 2401 Security Architecture for IP. IETF (1998)
17. Liberatore, M., Levine, B.N.: Inferring the Source of Encrypted HTTP Connections. In: Proceedings of the 13th ACM Conference on Computer and Communications Security (CCS 2006), pp. 255–263 (October 2006)
18. McLachlan, J., Hopper, N.: On the risks of serving whenever you surf: Vulnerabilities in Tor's blocking resistance design. In: Proceedings of the Workshop on Privacy in the Electronic Society (WPES 2009). ACM (November 2009)
19. Murdoch, S.J.: Hot or not: Revealing hidden services by their clock skew. In: Proceedings of CCS 2006 (October 2006)
20. Murdoch, S.J., Danezis, G.: Low-cost traffic analysis of Tor. In: Proceedings of the 2005 IEEE Symposium on Security and Privacy. IEEE CS (May 2005)
21. Murdoch, S.J., Zieliński, P.: Sampled traffic analysis by internet-exchange-level adversaries. In: Borisov, N., Golle, P. (eds.) PET 2007. LNCS, vol. 4776, pp. 167–183. Springer, Heidelberg (2007)
22. O'Connor, L.: On blending attacks for mixes with memory. In: Barni, M., Herrera-Joancomartí, J., Katzenbeisser, S., Pérez-González, F. (eds.) IH 2005. LNCS, vol. 3727, pp. 39–52. Springer, Heidelberg (2005)
23. Øverlier, L., Syverson, P.: Locating hidden servers. In: Proceedings of the 2006 IEEE Symposium on Security and Privacy. IEEE CS (May 2006)
24. Reed, M.G., Syverson, P.F., Goldschlag, D.M.: Anonymous connections and onion routing. IEEE Journal on Selected Areas in Communications 16(4), 482–494 (1998)
25. Rescorla, E.: SSL and TLS – Designing and Building Secure Systems. Addison-Wesley (2001)
26. Rybczyńska, M.: Network-level properties of modern anonymity systems. In: Proceedings of the International Multiconference on Computer Science and Information Technology, pp. 837–843 (2008)
27. Rybczyńska, M.: A round-based cover traffic algorithm for anonymity systems. In: 2009 International Conference on Intelligent Networking and Collaborative Systems, pp. 93–99 (2009)
28. Serjantov, A., Sewell, P.: Passive attack analysis for connection-based anonymity systems. In: Snekkenes, E., Gollmann, D. (eds.) ESORICS 2003. LNCS, vol. 2808, pp. 116–131. Springer, Heidelberg (2003)
29. Shmatikov, V., Wang, M.H.: Measuring relationship anonymity in mix networks. In: Proceedings of the Workshop on Privacy in the Electronic Society (WPES 2006) (October 2006)
30. Wang, M.-H.: Timing analysis in low-latency mix networks: Attacks and defenses. In: Gollmann, D., Meier, J., Sabelfeld, A. (eds.) ESORICS 2006. LNCS, vol. 4189, pp. 18–33. Springer, Heidelberg (2006)

31. Sun, Q., Simon, D.R., Wang, Y.M., Russell, W., Padmanabhan, V.N., Qiu, L.: Statistical identification of encrypted web browsing traffic. In: Proceedings of the 2002 IEEE Symposium on Security and Privacy, Berkeley, California (May 2002)
32. Wright, C.V., Monrose, F., Masson, G.M.: On inferring application protocol behaviors in encrypted network traffic. Journal of Machine Learning Research 7, 2745–2769 (2006)
33. Zalewski, M.: Silence on the Wire: a Field Guide to Passive Reconnaissance and Indirect Attacks. No Starch Press (2005)
34. Zhu, Y., Fu, X., Graham, B., Bettati, R., Zhao, W.: On flow correlation attacks and countermeasures in mix networks. In: Martin, D., Serjantov, A. (eds.) PET 2004. LNCS, vol. 3424, pp. 207–225. Springer, Heidelberg (2005)