Taxis: Scalable Strong Anonymous Communication

Andreas Hirt Michael Jacobson, Jr. Carey Williamson
Department of Computer Science
University of Calgary
{hirt,jacobs,carey}@cpsc.ucalgary.ca

Abstract

Anonymity protocols are a privacy-enhancing technology for Internet-based communication. Two important characteristics of anonymity protocols are the strength of anonymity provided, and the overhead required for anonymous communication. In this paper, we focus on the latter characteristic, and develop simple performance models for two anonymous communication protocols: Practical Buses and Taxis. We show that the message latency of the Practical Buses protocol scales quadratically with the number of participants, while that of the Taxis protocol scales linearly with the number of participants. Both models are validated with experimental measurements from prototype implementations of the protocols. We show that the Taxis protocol provides more scalable anonymous communication, without compromising the strength of anonymity provided.

1 Introduction

Anonymity protocols provide privacy for Internet-based communication, by protecting the identities of communicating parties from internal and external attackers.

There are numerous applications that require anonymity. For example, a Web server that provides controversial documents, such as democratic doctrine in a totalitarian regime, requires anonymity for freedom of speech and to prevent unjust prosecution. Military communication requires anonymity on the battlefield so that attackers cannot gain a tactical advantage by gleaning identity information from communications. Similarly, Internet-based counseling for victims of abuse requires anonymity in the early stages of the healing process. In general, anonymous communication enables freedom of speech, by eliminating the fear of retribution, risk, or ridicule that could occur if identities are compromised.

Anonymity can be classified into *data anonymity* and *connection anonymity*. Data anonymity [6] removes identifying data in messages, such as the sender address in an e-mail. Connection anonymity [6] obscures the traffic communication patterns. This prevents traffic analysis that traces a message through the network from the initiator to the responder. This paper focuses on *connection anonymity*.

Connection anonymity can be further divided into *sender anonymity*, *receiver anonymity*, *mutual anonymity*, and *unlinkability*. Sender anonymity [15] hides the identity of the initiator, while receiver anonymity [15] conceals the identity of the responder. Mutual anonymity [10] keeps the identities of both the initiator and responder anonymous. Unlinkability [15] prevents the linking of a message sent by an initiator to its responder, and vice versa. The type of connection anonymity required depends upon the application.

Two important characteristics of anonymity protocols are the strength of anonymity provided, and the overhead required for anonymous communication. There is typically a trade off between these two properties. For example, strong anonymity can be provided by aggregating many messages into batches (mixes) before forwarding them, but this adversely affects message delay. Similarly, anonymity can be strengthened with dummy cover traffic (i.e., fake messages) on the network, but this increases the bandwidth consumption as well as the processing overhead for each participating node. Reducing the volume of cover traffic may compromise anonymity, but it makes the protocol vulnerable to identity-discerning attacks. Finding a low-overhead anonymous communication scheme with provably strong anonymity is a challenge.

In this paper, we focus on the performance of anonymity protocols. In particular, we consider the Buses protocol from the literature: a known protocol with provably strong anonymity properties. We develop a simple performance model for the Practical

Buses protocol, an extension to the Buses Protocol, and show that the message latency scales quadratically with the number of participants in the network. We then propose an improved protocol called Taxis, and show that its message latency scales linearly with the number of participants. Both models are validated with experimental measurements from a prototype implementation of each protocol. Furthermore, we show that Taxis provides more scalable anonymous communication than Practical Buses, without compromising the anonymity provided.

The rest of this paper is organized as follows. Section 2 provides background information on anonymity protocols and related work. Section 3 discusses the Practical Buses protocol and presents an analysis of its average message latency. Section 4 describes the new Taxis protocol, while Section 5 shows that its average message latency scales linearly. Section 6 presents experimental results, while Section 7 shows that Taxis retains the strong anonymity guarantees of Practical Buses. Finally, Section 8 concludes the paper.

2 Background and Related Work

2.1 Anonymity Techniques

Anonymous communication is a vibrant research field with many proposed anonymous communication schemes [1]. Applications such as e-mail have incorporated strong anonymous communication schemes, since they are delay-tolerant [7]. However, interactive applications, such as Web browsing and SSH, require lower latency. Designing a strong anonymous communication scheme with low overhead is an open research problem [8]. As an interim solution, low-latency anonymous communication schemes, such as Crowds [17] and Tor [8], have been used for interactive applications, despite being vulnerable to known attacks [8].

The three basic anonymity techniques are *broadcasting*, *mixes*, and *buses*. Anonymous communication protocols inherit their performance and anonymity strength from these underlying techniques.

The broadcasting technique requires each participant to broadcast to all the other participants the parities of its secret coin flips [5]. Since this overhead scales quadratically, the broadcasting approach is no longer mainstream in the literature [1], and thus is not considered further in this paper.

The mixes and buses techniques can be measured by their performance and the strength of anonymity provided. The performance metric considered in this paper is the average message latency, and how it scales with the number of participants. The measure of anonymity is how well an initiator or responder is hidden among the participants.

To measure the strength of anonymity, the nomenclature *anonymity set*, *sender anonymity set*, and *receiver anonymity set* are used [14]. The anonymity set consists of all of the participants in an anonymous communication scheme. The sender anonymity set is a subset of the anonymity set that contains all of the participants that could have been an initiator for a particular message. Similarly, the receiver anonymity set is a subset of the anonymity set that contains all of the participants that could have been a responder for a message. A desirable characteristic of strong anonymity is that both the sender anonymity set and receiver anonymity set are equal to the anonymity set.

2.2 Mixes

In 1981, Chaum published his landmark paper on mixes. This was the first anonymity technique published and it provides very strong anonymity. The idea is to re-route a multiply encrypted message through a sequence of mixes. Each mix collects a batch of inputs, peels away a layer of encryption from each input, and randomly re-orders the outputs.

The goal is to prevent an attacker from correlating an input to its corresponding output, assuming that the mix's private key is kept confidential. However, the events of an initiator sending a message and a responder receiving a message also need to be hidden. The only solution proved to be secure requires each initiator to send at least one message to each responder in every single batch. This *full cover traffic* hides legitimate messages among the dummy messages. Uniform message size and replay protection must also be provided to preserve anonymity [4].

The problem with mixes is that the cover traffic scales quadratically with the number of participants. There is some debate as to how much anonymity is sacrificed by reducing the cover traffic [8]. Some schemes use no cover traffic, partial cover traffic, or almost full cover traffic. However, none of these solutions are 100% secure against a dedicated adversary, and few are suitable for interactive applications.

Anonymous communication schemes such as Tor [8] and Crowds [17] use no cover traffic at all. As a result, no batches are collected for input-output mixing. The authors of both these schemes acknowledge that they achieve their low overhead at the cost of being vulnerable to known attacks [8, 17].

Schemes such as JAP [3] and MorphMix [18] use partial cover traffic. JAP has each client send a constant stream of dummy traffic through a cascade of mixes,

but no dummy traffic is sent to the receivers. Thus, the intersection attacks discussed in [3] can be expedited with a traffic confirmation attack. An adversary corrupts each suspected sender's stream of traffic before the mix cascade, one at a time, and looks for a corresponding lack of traffic at the receiver. MorphMix is based on a peer-to-peer design that creates partial cover traffic through the construction of its anonymous tunnels. However, colluding peers can circumvent its collusion detection mechanism and defeat anonymity for a significant fraction of the communications [19].

Mixmaster [13], Mixminion [7], and Tarzan [9] use almost full cover traffic. Mixmaster and Mixminion only use internal dummy traffic among the mixes, but not between the initiators or responders. Both are suitable only for delay-tolerant applications such as e-mail because of their high overhead in collecting batches of sufficient size. Tarzan has mimics exchange cover traffic with a subset of the network (e.g., with six other peers). However, it requires each peer to have accurate knowledge of the entire topology. With dynamic membership, such as on the Internet, the routing protocol can be quickly overwhelmed with updates.

2.3 Buses

In 2003, Beimel and Dolev introduced the buses anonymity technique [2]. The optimal buffer version is based upon a metaphorical city bus, which traverses all of the participants in the anonymity network. The key idea is that messages can travel in the seats of a bus, appropriately obscured by layers of encryption. The bus traverses the network, passing from one node to the next. When in possession of the bus, a node replaces the bus with its decryption and retrieves messages intended for it, replacing the message's seat with random data. In addition, the node places layered encrypted messages on the bus to be delivered to other nodes. By encrypting the message, only the intended recipient can recover its messages. By having each node decrypt the bus, an adversary cannot tell if a node replaces a seat with its decryption or inserts a new message to be sent.

3 The Practical Buses Protocol

3.1 Protocol Overview

The Practical Buses protocol [11, 12] extends buses [2] to improve its security and performance in practice. Each participant owns a 'row' of bus seats, which it uses to send and forward messages. To hide the event of a node replacing a seat with a message to send or forward, each node replaces all of its owned seats each time it receives the bus with either random data or a valid hybrid encrypted message (using RSA-OAEP and AES-CBC). There are no known techniques that an adversary can use to distinguish valid messages from random data, unless it possesses the corresponding decryption key.

When a participant receives the bus, it copies the bus, modifies the bus, and forwards the bus. The bus is copied for off-line processing, which can take place after the incoming bus has been forwarded. The main component of the off-line processing is to decrypt the bus, removing a layer of encryption from any nested messages with the correct OAEP redundancy. A participant modifies the bus by inserting any messages to forward, then any messages to send, and finally replacing any unused owned seats with random data, before forwarding the bus to the next participant on the route.

A new *nested encrypted message* is created by (hybrid) encrypting a message with the responder's key, selecting a random set of up to l indirection participants (e.g., $l = 2$), and then recursively encrypting the message with each indirection participant's corresponding key. This resulting indirection path requires at least one and at most $l+1$ complete bus tours of all the participants to deliver the message to the responder (the 1 arises from the fact that the recipient could be at the end of the bus tour).

Practical Buses improves both the performance and security of Buses [2] (see [11, 12] for details). Performance is improved by using expired seats, persistent TCP connections, and multi-threading. Security is improved via anonymous acknowledgements, seat signatures, and replay protection.

In Buses and Practical Buses, the bus path creates an artificial re-routing path for a message, just like mixes. However, unlike Mixes, they do not require quadratic cover traffic for strong anonymity, nor do they require mixing. In addition, a participant that handles a nested encrypted message can only identify the predecessor in the indirection path, while the successor is unknown.

Practical Buses provides strong mutual anonymity. It is not vulnerable to any known attacks, except DoS attacks [11]. The sender anonymity set and receiver anonymity set both consist of the full anonymity set.

3.2 Message Latency Analysis

The goal of our analysis of the Practical Buses protocol is to model the average message latency: the elapsed time from when a message is entered to when the anonymous acknowledgement is received and processed. On average, the time until the responder re-

ceives the message is half the average message latency. For simplicity, we assume that the network is reliable, and messages do not have to be resent.

The average message latency, $E[L_B]$ is calculated as:

$$E[L_B] = 2(d_{IR} + h \underbrace{(d_{net} + d_{proc})}_{\text{per hop}}) \quad (1)$$

where d_{IR} denotes the sending delay to construct a nested encrypted message (I) and the delay to process a received message offline (R), h denotes the average number of bus hops to deliver a message, d_{net} denotes the average bus network delay per hop, and d_{proc} denotes the average bus processing delay per hop. The factor of 2 accounts for the message being sent and the anonymous acknowledgement being received.

The average number of bus hops between a message being sent and received depends on how far away the bus is from the initiator when it wants to send a message (h_w), the number of hops (h_{IR}) along the bus route between the initiator and the responder, the location of the bus nodes for each of the l indirection nodes, and the offline processing for each of the l indirection nodes that causes a forwarded message to be delayed an entire bus tour. Let the bus path consist of n nodes. On average $h_w = (n-1)/2$ and each layer of encryption requires an additional bus tour with probability 1/2 due to the location of the forwarding node/responder. For simplicity, assume that a message to send/forward is not delayed by other messages to send or forward. The average number of hops to deliver a message is:

$$h = h_w + n * \underbrace{\left(\frac{h_{IR}}{n} + \frac{l}{2} + l\right)}_{\text{\# of bus tours}} = \frac{3nl + 2n - 1}{2} \quad . \quad (2)$$

The average network delay per bus hop depends on the transmission delay (d_{xmit}), propagation delay, processing delay, queueing delay, and TCP delay. For simplicity, let propagation delay, processing delay, and queueing delay be approximated by a constant c. In addition, assume that TCP setup delay can be ignored since persistent connections make this delay a one-time cost. If r denotes the transmission rate in bits per second, k the number of seats per node, n the number of nodes, s the seat size in bits, then $d_{xmit} = kns/r$. Thus, the average network delay can be calculated as

$$d_{net} = \frac{kns}{r} + c \quad . \quad (3)$$

The average processing delay per bus hop depends on the total number of seats on the bus (kn) and the average delay to process a seat (d_{seat}) :

$$d_{proc} = knd_{seat} \quad . \quad (4)$$

Substituting (2), (3), and (4) into (1) and simplifying yields the average message latency for Practical Buses. $E[L_B]$ equals

$$2\left(d_{IR} + \left(\frac{3nl + 2n - 1}{2}\right)\left(\left(\frac{kns}{r} + c\right) + knd_{seat}\right)\right) \quad (5)$$

where d_{IR} is the average delay to construct a new nested encrypted message or anonymous acknowledgement (I) plus the average delay caused by the offline processing of a received message or anonymous acknowledgement (R), l is the average number of indirection participants, n is the number of participants in the Practical Buses network, k is the number of seats owned by each participant in the Practical Buses network, s is the size of each seat on the bus, r is the transmission rate, c represents the sum of the queueing, processing, and propagation network delays, and d_{seat} is the average delay to process a seat.

According to our analysis, the message latency of the Practical Buses protocol is $O(n^2)$, scaling quadratically with the number of participants. Experimental results from a prototype implementation confirm this observation (see Section 6).

4 The Taxis Protocol

4.1 Motivation

Quadratic scalability of the message latency makes the 'Practical' Buses protocol impractical. Equation (5) shows that the quadratic dependency arises from the number of hops to deliver a message multiplied by the processing delay and network delay, each of which are $O(n)$. However, the analysis of the Practical Buses protocol provides the insight required to develop a better protocol, which we call Taxis.

The key idea in Taxis is to reduce both the processing and network delay per tour. Instead of a bus with rows of owned seats, each participant owns a single taxi that contains only its owned seats. This allows a participant to separate the taxis into owned or unowned taxis. The unowned taxis require no on-line processing, and can simply be forwarded, resulting in a metaphorical 'fast lane'. In this way, a taxi is only delayed once per tour for on-line seat processing at the participant that owns the taxi. To provide further insight into the taxis, the message life cycle and taxi processing steps are discussed next.

4.2 Message Life Cycle

The creation of the initial nested encrypted message involves choosing a random indirection path of random

but bounded length. The inner core holds the actual message, an anonymous public key (required for the receiver to send an anonymous acknowledgement) and a message tag field. When adding a layer of encryption either for the receiver or an indirection node, a random AES key in the salt field and a forward flag (true for every layer except the inner core) are prepended to the current message content. The salt is encrypted with RSA-OAEP and the rest of the message is encrypted with AES using the salt field value as the key.

Encapsulation in a seat consists of prepending the nested encrypted message with the seat owner's public key, seat tag (unique seat identifier), and seat timestamp, and appending random data to make the seats uniform in size. Then a signature, calculated on the public key, seat tag, timestamp, encrypted nested message, and random data, is appended to the seat.

A node processing a nested encrypted message first tries to extract the AES key by decrypting the salt field with both its RSA-OAEP private and anonymous private key. This decryption is successful if the result has the prescribed redundancy, in which case the node removes the outermost encryption layer by decrypting the rest of the data with the extracted AES key. After decrypting, the forward field is extracted. If the forward flag is true, then it removes the salt (and the forward fields) and forwards the rest of the decrypted inner layer. If the forward flag is false, the node extracts the actual message from the inner core.

4.3 Taxi Processing

The processing of a taxi consists of receiving, modifying, and forwarding the taxi. The taxi that a participant does not own is always received and then immediately forwarded using the 'fast lane', with the taxi processed off-line thereafter. On the other hand, an owned taxi is received, modified, and then forwarded. An owned taxi always has a lower priority so that the fast lane is not delayed by the processing delay of owned seats. The receiving, modifying, and forwarding of the taxi are now discussed in further detail.

Receiving a taxi consists of copying it for the off-line extraction of messages. For each decrypted salt field with the prescribed redundancy, the AES key is extracted and the rest of the seat is decrypted. The signature for the seat is checked. If it fails, then the contents of the decrypted seat are not further processed; otherwise, it is processed. If the forward flag is true, then the contents of the decrypted message are inserted into the *send queue* for transmission at the next available opportunity. If the forward flag is false, then the contents of the decrypted seat are passed up to the application layer. Incoming messages trigger an anonymous acknowledgement, which is created and placed in the send queue.

Modifying an owned taxi with k seats consists of replacing all of the seats with either new messages being sent, message forwarding, the sending of anonymous acknowledgements, messages being resent, or random data.

Forwarding a taxi consists of sending it to the next node in the taxi path. A participant sends and forwards messages only in its owned taxi. Nested encrypted messages are inserted into the seats of a taxi in an identical manner as Practical Buses, with an indirection path and recursive hybrid encryption.

5 Analysis of Taxis Protocol

5.1 Performance Advantages

Separating taxis into owned or unowned taxis has two important performance advantages:

- *Improved parallelism.* $O(n)$ participants send a taxi each time unit, so the network is fully pipelined. This effectively reduces network delay to that of the slowest link on the bus path. Unowned taxis have a higher precedence over owned taxis. This creates a metaphorical fast lane for unowned taxis, which can be forwarded immediately after they are copied for local off-line processing.

- *Reduced processing delay.* Due to the segregation of taxis into two classes, the processing delay is reduced by a factor of $O(n)$. As a result, the processing delay is reduced to the time it takes for the slowest participant to modify its taxi.

The improved parallelism can be explained the following way. If there is a 'train' of q taxis that arrives at a participant, then the size of the departing train is $q-1$ if the participant owns one of the taxis, and q otherwise. To see this, observe that if a node owns a taxi in the train, all of the unowned taxis have higher priority and are forwarded first. Then the owned taxi is then processed before it is forwarded, which separates it from the train of unowned taxis.

The main result is that Taxis reduces the network delay of Practical Buses by a factor of n. To understand this, let one time unit represent the time it takes for a taxi to traverse one link, p_i denote the i^{th} participant, p_{i+1} denote the next participant in the static round-robin taxi path, t_i denote the taxi owned by p_i, and n denote the total number of participants.

Consider the worst case scenario, where one of the participants has had all of the taxis arrive at the same time to form one long train at p_i. Without loss of generality, assume that this train arrives at p_1 at time 1. Since n taxis arrived at p_1, the owned taxi t_1 must also be received by p_1. By the earlier argument, the outgoing train has a length of $n-1$, and taxi t_1 is delayed. Thus, a train is sent containing the taxis $t_2, t_3, \ldots, t_n$ in the order they arrived. By the second time unit, p_2 receives the taxi train. Again, p_2 delays its taxi t_2, and forwards the taxi train containing the taxis $t_3, t_4, \ldots, t_n$ in the order they arrived. In addition, the delayed taxi t_1 is sent during time unit 2. Using the same argument and induction, p_i receives a train of size $(n-i+1)$ containing the taxis $t_i, t_{i+1}, ..., t_n$ at time unit i. It delays the taxi t_i, forwards a train of size $(n-i)$, and $(i-1)$ taxis are forwarded by previous participants that delayed their taxis. The average number of taxis $E[T]$ sent per time unit (not including the train) over one full tour $i = 1, 2, \ldots, n$ is:

$$E[T] = \frac{\sum_{i=1}^{n}(i-1)}{n} = \frac{n-3}{2} \quad ,$$

which is $O(n)$. Therefore, the network delay of Taxis is reduced by a factor of n compared to Practical Buses.

Taxis reduces the processing delay of Practical Buses by a factor of $O(n)$. Consider Practical Buses where there are n participants and a bus makes one full tour, visiting all the participants exactly once. The bus is delayed at each participant so that the participant can process its row of owned seats. Thus, each row of owned seats is delayed $\sum_{i=1}^{n} d_{proc_i}$ per bus tour, where d_{proc_i} is the delay induced by a participant p_i processing its owned seats. This has the unfortunate consequence that each participant delays the unowned seats unnecessarily.

Taxis does not delay unowned seats unnecessarily. Recall that taxis are segregated into two classes, owned and unowned taxis. Consider the full tour of a taxi starting at its owner p_i. Before p_i forwards its taxi, the taxi is first modified. Then p_i forwards the taxi to complete a full tour of the network. Since no other participant owns the taxi, it will travel through a metaphorical 'fast lane' during this full tour, and is only copied for offline processing by each participant. The taxi is able to complete a full tour of the network very quickly, and is only delayed by the owner for d_{proc_i} for processing. Similarly, all the other taxis incur one processing delay per full tour. As a result, all the taxis tour the network with a processing delay of d_{proc_i}. Therefore, the processing delay of Taxis is reduced by a factor of $O(n)$ compared to Practical Buses.

5.2 Message Latency Analysis

The formula to model the average message latency of Taxis is built upon the formula to model that of Practical Buses in Section 3.2. The average number of hops required to deliver a message is still the same as in the Practical Buses model (see Equation (2)). However, the average time it takes a participant to send all the taxis and process all the taxis on average per hop has changed.

Recall that the network delay is reduced by a factor of n, and that the processing delay is reduced by a factor of $O(n)$. Dividing the respective Practical Buses model Equations (3) and (4) by n yields:

$$d_{net} = \frac{ks}{r} + \frac{c}{n} \tag{6}$$

and

$$d_{proc} = kd_{seat} \quad . \tag{7}$$

Substituting Equations (2), (6), and (7) into the message latency model for the Practical Buses protocol Equation (1) yields the following model for Taxis. $E[L_T]$ equals

$$2\left(d_{IR} + \left(\frac{3nl+2n-1}{2}\right)\left(\left(\frac{ks}{r} + \frac{c}{n}\right) + kd_{seat}\right)\right) \tag{8}$$

using the same notation as defined previously.

The number of hops required is still $O(n)$, but the network and processing delay are both reduced by $O(n)$. Therefore, Taxis' average message latency scales linearly with the number of participants.

6 Experimental Results

6.1 Experimental Methodology

We developed prototype implementations of the Practical Buses protocol and the Taxis protocol, and evaluated them experimentally. The protocols are tested on a LAN consisting of a Beowulf cluster of 14 identically configured machines. Each machine is a dual-processor 2.4 GHz machine with 2 GB RAM and a 512 KB cache. The machines are connected by a 1 Gbps Ethernet switch. The software used is Linux 2.4.20-19.7, POSIX threads with libc-2-2.5, GMP Version 4.1.1, and g++ version 3.2. To simplify the analysis, a statically-configured round-robin tour of the participating nodes is used with persistent TCP connections. In addition, only a single CPU is used on the dual-processor machines.

A Practical Buses or Taxis peer program is run on each participating node in the LAN. The peer program

uses a TCP overlay to provide anonymous communication for a chat application written specifically for testing. The chat program uses pre-defined workload files, so that the exact same workload (i.e., message sizes and timings) is used for each anonymity protocol.

For consistency, the experimental results for the Taxis protocol and the Practical Buses protocol used the same basic implementation. The design of the Taxis protocol is very similar to the design of the modified Practical Buses in Section 2.3. The changes consisted of dividing the bus into taxis, incorporating a fast lane, and adding in the handling of taxis.

The primary performance metric is message latency. It is extracted from trace files recorded by the peer program and calculated by taking the difference between when the bus or taxi peer first receives a message from the chat application and when the chat program receives the corresponding anonymous acknowledgement.

The testing consisted of scaling from 2 to 14 participants. For each number of participants, a total of 360 messages with a random length between 1 byte and 2 KB are exchanged. A Poisson message arrival process with an aggregate average of 30 messages per minute is used. The initiators and responders of messages are randomly chosen. A Zipf distribution is approximated by having two "elephant" participants generate 50% of the traffic when there are more than four participants. The seat size is 3 KB, and each participant owns a two-seat taxi. The number of indirection nodes is randomly chosen as either 1 or 2.

6.2 Model Validation Results

Figure 1 shows the experimental results for round-trip message latency, for both Practical Buses and Taxis. The average message latency increases from about 0.5 seconds to several seconds as the number of participants increases from 2 to 14. The Taxis protocol exhibits lower latency than the Practical Buses protocol.

Based on our earlier analysis, the message latency for Practical Buses scales quadratically with the number of participants. The experimental results confirm this analysis (see Figure 2). The curve fitting is done using the non-linear least-squares (NLLS) Marquardt-Levenberg algorithm in gnuplot. The fitting parameter values for d_{IR} and d_{seat} are also shown in Figure 2. The model fits the experimental results well.

For the Taxis protocol, the average message latency scales linearly with the number of participants. The modeling results for the Taxis protocol are fit to the experimental results in Figure 3. The linear model fits the experimental results well.

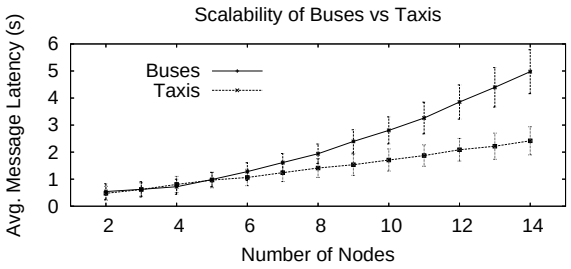


Figure 1. Experimental results for average message latency for Practical Buses and Taxis

7 Security Analysis of Taxis

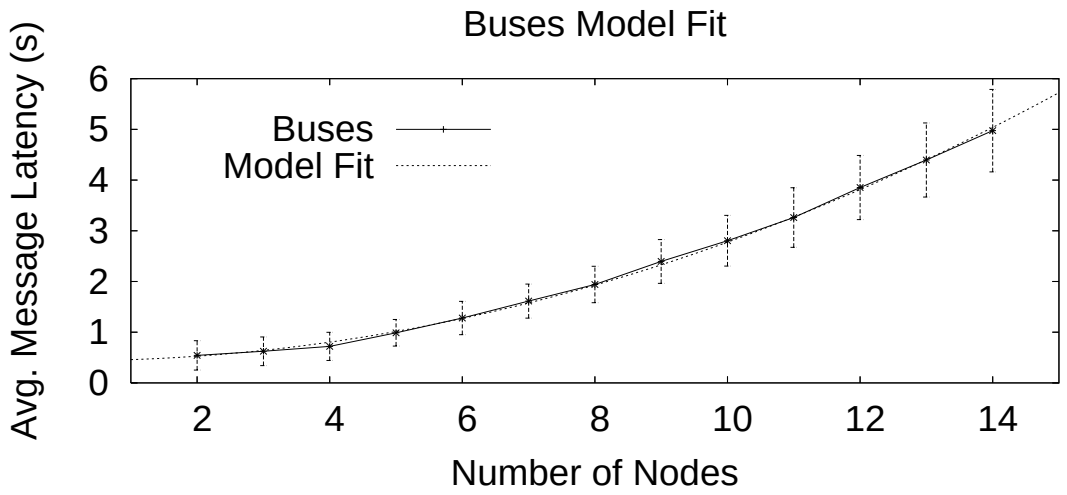
While the foregoing experimental results demonstrate that the Taxis protocol is significantly faster than the Practical Buses protocol, we still need to verify that anonymity has not been compromised. That is, due to the significant changes in the Taxis protocol, it is necessary to re-evaluate its security.

The Practical Buses protocol is analyzed in [12], and shown to be secure against all attacks, except DoS attacks. The threat model used for the analysis consists of an eavesdropper, passive adversary, or active adversary. In particular, an eavesdropper can view anything exchanged between two participants on the network. A passive adversary can corrupt one or more participants to relate incoming messages to their corresponding outgoing messages, as well as view the routing tables and decryption keys of the corrupted participants. An active adversary has all the powers of a passive adversary, plus the ability to create, modify, and delete messages.

The goal of an eavesdropper, passive adversary, or active adversary is to collect observable events in order to reduce anonymity. An active adversary also tries to create additional observable events that can be used to reduce anonymity.

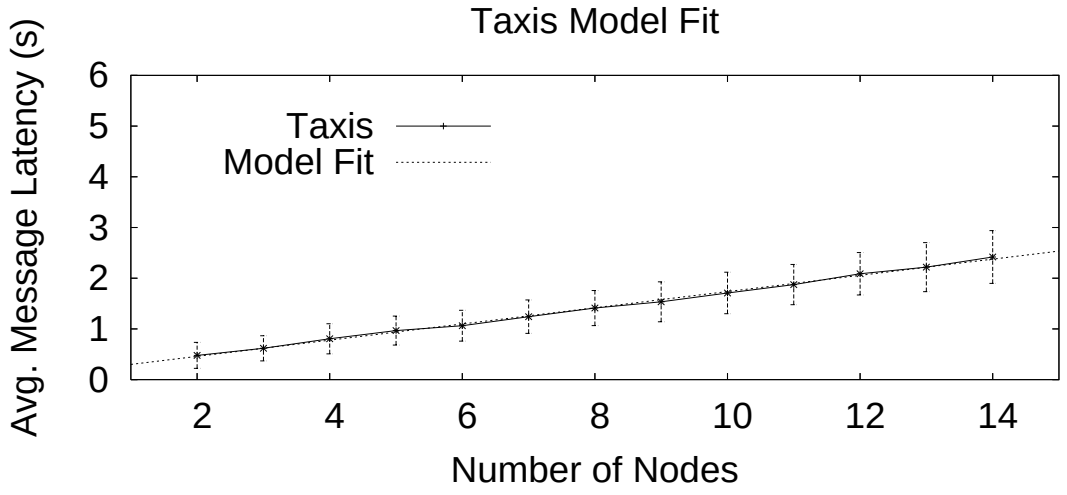
The Practical Buses protocol is not vulnerable to any known attacks including local eavesdropper before/after proxy, size/time correlation, low load, full compromised path, marker, passive traceback timing, replay, spam, mob, filter, and intersection attack [11]. The Taxis protocol is also not vulnerable to any of these attacks. A simple analysis shows that the Taxis protocol has the necessary counter-measures listed in [11] that are required to defeat the attack.

However, the Taxis protocol may be vulnerable to new types of attacks, and warrants further analysis. Recall that the adversary's goal is to collect or create



$k = 2$ seats, $s = 24,576$ bits, $l = 1.5$ layers, $r = 1$ Gbps, $c = 0.00001$ seconds
$d_{s+r} = 0.219597 \pm 0.009779$ seconds $\quad$ $d_{seat} = 0.00181356 \pm 1.538e^{-05}$ seconds

Figure 2. Quadratic fit of Buses model to experimental results.



$k = 2$ seats, $s = 24,576$ bits, $l = 1.5$ layers, $r = 1$ Gbps, $c = 0.00001$ seconds
$d_{s+r} = 0.0819427 \pm 0.009394$ seconds $\quad$ $d_{seat} = 0.0122894 \pm 0.0001663$ seconds

Figure 3. Linear fit of Taxis model to experimental results.

observable events in order to reduce anonymity. Thus, the focus of this analysis is based upon identifying and analyzing any new information observable within the threat model.

The new observable event in Taxis is the sending and receiving of individual taxis. An adversary can try to glean information from passively observing the taxis that circulate the network under normal conditions, or can create observable events by delaying, modifying, or deleting messages or taxis.

7.1 Passive Attacks

The adversary monitors the taxis that circulate the network, recording when they stop at each of the participants. It can then try to leverage this information to reduce either sender or receiver anonymity.

An adversary can try to reduce sender anonymity by observing when a participant sends a valid message through an adversary to forward. The adversary has to then determine if the predecessor is an initiator to defeat sender anonymity. This requires that the adversary eliminate the possibility that no other participant sent the suspected initiator a message to forward. As a result, any participants who exchange taxis with the suspected initiator could have forwarded the message through the suspected initiator. Clearly, if the taxis are regularly circulating the network, the sender and receiver anonymity sets are not reduced.

An adversary can also try to reduce receiver anonymity. However, this is not feasible since an adversary cannot determine the predecessor in the indirection path.

7.2 Active Attacks

An active adversary could try to add, modify, or delete taxis in order to reduce anonymity. The modifying and adding of taxis requires the adversary to steal the corresponding private key(s). Otherwise, the signatures on the seats will be invalid, and the spoofed seats will be ignored.

Adding a taxi does not defeat anonymity. The only noticeable events that occur are at the corrupted participants, who may receive the added message(s), remove a layer of encryption, and forward it or potentially receive it. However, this does not create any new events that reduce anonymity.

Modifying a taxi does not defeat anonymity. Any subsequent layers of encryption will not decrypt successfully, since the required redundancy for an OAEP decryption to succeed is destroyed. The only noticeable events occur at the corrupted participants, who may receive the message, attempt to remove a layer of encryption, and then give up when the OAEP redundancy check fails. As a result, no new events are created that help to reduce anonymity.

The deletion of a taxi does create an observable event, since a participant does not receive the taxi on a regular schedule. Thus, the sender anonymity sets for all the participants are reduced when the taxi is not circulating. However, this is analogous to a temporary DoS attack and all anonymous communication schemes either have their anonymity set reduced under such conditions or stop sending messages. Thus, the attacker can effectively reduce anonymity. The participants are quickly made aware of this attack, since the lack of anonymous acknowledgements generated by messages being sent or forwarded on the deleted taxis are noticeable by the participants, in addition to the owner timing out on its own taxi being received.

Thus, the only attack that an active adversary can use to defeat anonymity is to try to delete multiple taxis and reduce the anonymity sets. In the best case, the adversary must repeatedly delete all of the taxis before they are received by any other participant, except the initiator's taxi, the responder's taxi, and the indirection participants' taxis that are used to forward the message. This reduces the anonymity set to its minimum $(l + 2)$, where l is the number of indirection nodes. However, this presents a paradox for the adversary, who must successfully guess the initiator, responder, and indirection participants within a short time window before being detected. (Note that the attacker does not know the random indirection path before the attack is launched.) The number of indirection paths for Taxis with n participants and l indirection nodes is n^l. Thus, as the size of the network grows, it becomes difficult for the adversary to guess the indirection path successfully. The same paradox arises when the attacker uses a combination of corrupted participants, and deleted taxis.

Thus, the Taxis protocol maintains the same anonymity strength as Practical Buses.

8 Conclusions

In this paper, we focus on the performance of anonymous communication protocols, and develop latency models for Practical Buses and Taxis. We show that the message latency of the Practical Buses protocol scales quadratically with the number of participants, while that of the Taxis protocol scales linearly with the number of participants. Both models are validated with experimental measurements from prototype implementations of the protocols. Furthermore, we show

that the improved performance of the Taxis protocol does not compromise the strength of anonymity provided by the Practical Buses protocol.

While the Taxis protocol is promising, possible extensions could further reduce its overhead. Currently, the number of hops to deliver a message scales linearly because each taxi visits every participant in the anonymity network. This linear dependence could be reduced by transferring seats between taxis that only traverse a subset of the network; such a hierarchy could result in logarithmic scalability. The processing delay of the Taxis protocol could also be significantly reduced. One possible solution is to use (slow) public-key cryptography taxis to exchange symmetric keys with potential indirection participants, enabling (fast) symmetric-key cryptography taxis for actual messages. We are currently implementing these extensions, and plan to test them on a larger scale using PlanetLab [16].

References

[1] Anonymity bibliography, 2005. `http://www.freehaven.net/anonbib/`.

[2] A. Beimel and S. Dolev. Buses for anonymous message delivery. *Journal of Cryptology*, 16(1):25–39, 2003.

[3] Oliver Berthold, Hannes Federrath, and Stefan Köpsell. Web MIXes: A system for anonymous and unobservable Internet access. In H. Federrath, editor, *Proceedings of Designing Privacy Enhancing Technologies: Workshop on Design Issues in Anonymity and Unobservability*, pages 115–129. Springer-Verlag, LNCS 2009, July 2000.

[4] D. Chaum. Untraceable electronic mail, return addresses and digital pseudonyms. *Communications of the ACM*, 24(2):84–88, 1981.

[5] D. Chaum. The dining cryptographers problem: Unconditional sender and recipient untraceability. *Journal of Cryptology*, 1(1):65–75, 1988.

[6] J. Claessens, B. Preneel, and J. Vandewalle. Solutions for anonymous communication on the internet. In *Proceedings of the International Carnahan Conference on Security Technology*, pages 298–303. IEEE, 1999.

[7] G. Danezis, R. Dingledine, and N. Mathewson. Mixminion: Design of a type III anonymous remailer protocol. In *Proceedings of the 2003 IEEE Symposium on Security and Privacy*, May 2003.

[8] R. Dingledine, N. Mathewson, and P. Syverson. Tor: The second-generation onion router. In *Proceedings of the 13th USENIX Security Symposium*, August 2004.

[9] M. Freedman and R. Morris. Tarzan: A peer-to-peer anonymizing network layer. In *Proceedings of the 9th ACM Conference on Computer and Communications Security (CCS 2002)*, Washington, DC, November 2002.

[10] Y. Guan, X. Fu, R. Bettati, and W. Zhoa. An optimal strategy for anonymous communication protocols. In *Proceedings of 22nd International Conference on Distributed Computing Systems*, pages 257–266. IEEE, 2002.

[11] A. Hirt. A practical buses protocol for anonymous network communication. Master's thesis, University of Calgary, Calgary, Alberta, August 2004.

[12] Andreas Hirt, Jr. Michael Jacobson, and Carey Williamson. A practical buses protocol for anonymous Internet communication. In *Third Annual Conference on Privacy, Security, and Trust*, pages 233–236, October 2005.

[13] Ulf Möller, Lance Cottrell, Peter Palfrader, and Len Sassaman. Mixmaster Protocol — Version 2. Draft, July 2003.

[14] A. Pfitzmann and M. Köhntopp. Anonymity, unobservability, and pseudonymity: A proposal for terminology. Draft, version 0.31, February 2008.

[15] A. Pfitzmann and M. Waidner. Networks without user observability. *Computers & Security*, 2(6):158–166, 1987.

[16] Planetlab, 2006. `http://www.planet-lab.org`.

[17] M. Reiter and A. Rubin. Crowds: Anonymity for web transactions. *ACM Transactions on Information and System Security*, 1(1):66–92, 1998.

[18] M. Rennhard and B. Plattner. Introducing morphmix: Peer-to-peer based anonymous internet usage with collusion detection. In *Proceedings of the Workshop on Privacy in the Electronic Society (WPES 2002)*, Washington, DC, USA, November 2002.

[19] P. Tabriz and N. Borisov. Breaking the collusion detection mechanism of Morphmix. In G. Danezis and P. Golle, editors, *Proceedings of the Sixth Workshop on Privacy Enhancing Technologies 2006*, pages 368–384. Springer, June 2006.