# Introducing MorphMix: Peer-to-Peer based Anonymous Internet Usage with Collusion Detection

Marc Rennhard
Swiss Federal Institute of Technology
Computer Engineering and Networks Laboratory
Zurich, Switzerland
rennhard@tik.ee.ethz.ch

Bernhard Plattner
Swiss Federal Institute of Technology
Computer Engineering and Networks Laboratory
Zurich, Switzerland
plattner@tik.ee.ethz.ch

## ABSTRACT

Traditional mix-based systems are composed of a small set of static, well known, and highly reliable mixes. To resist traffic analysis attacks at a mix, cover traffic must be used, which results in significant bandwidth overhead. End-to-end traffic analysis attacks are even more difficult to counter because there are only a few entry- and exit-points in the system. Static mix networks also suffer from scalability problems and in several countries, institutions operating a mix could be targeted by legal attacks. In this paper, we introduce MorphMix, a system for peer-to-peer based anonymous Internet usage. Each MorphMix node is a mix and anyone can easily join the system. We believe that MorphMix overcomes or reduces several drawbacks of static mix networks. In particular, we argue that our approach offers good protection from traffic analysis attacks without employing cover traffic. But MorphMix also introduces new challenges. One is that an adversary can easily operate several malicious nodes in the system and try to break the anonymity of legitimate users by getting full control over their anonymous paths. To counter this attack, we have developed a collusion detection mechanism, which allows to identify compromised paths with high probability before they are being used.

## Categories and Subject Descriptors

C.2.0 [**Computer-Communication Networks**]: General—*Security and protection*; C.2.4 [**Computer-Communication Networks**]: Distributed Systems—*Distributed applications*

## General Terms

Algorithms, Design, Measurement, Security

## Keywords

anonymity, mix networks, peer-to-peer systems, collusion detection


*WPES'02*, November 21, 2002, Washington, DC, USA.


## 1. INTRODUCTION

In 1981, David Chaum proposed the concept of a *mix network* [5], which is considered as the most promising approach to solve the problem of anonymous communication in the Internet. Since then several systems based on Chaum's idea to provide anonymous access to Internet services have been operational. The mixmaster system [6] follows Chaum's original design closely and enables users to send and receive (using reply blocks) electronic mail anonymously. Variations to the basic mix design to support near-real-time services such as web browsing have led to circuit-based systems: Onion Routing [14], Freedom [4], Web Mixes [2], and the Anonymity Network [17]. Most mix-based systems offer *sender* and *relationship anonymity* [13]. Although there are applications for receiver anonymity such as anonymous web publishing [22], most Internet activities where anonymity is desired require only sender and relationship anonymity.

Usually, mix networks consist of relatively few and well-known mixes. To communicate anonymously with a server in the case of a circuit based mix network, a user establishes an anonymous path via a subset of the mixes. The mixes relay all traffic exchanged between the user's computer and the server along this path. To be resistant against traffic analysis attacks, a mix network employs fixed-length messages and layered encryption of messages. In addition, mixes delay and reorder incoming messages from different users and use cover traffic to hide real messages. Finally, each mix processes each message only once to counter replay-attacks.

Traditional mix networks offer several benefits: the mixes' IP addresses can be made public through web sites or the Usenet, which allows accessing them easily. Digital certificates [12] allow to control which mixes offer their services, which makes it difficult for unauthorized (and potentially malicious) mixes to join. In addition, by controlling who is allowed to operate a mix, one can make sure that only highly reliable mixes with lots of computing power and good network connectivity are present in the system.

On the other hand, there are several limitations: the number of mixes is relatively small compared to the potential number of users, which implies the system eventually reaches its limits with respect to the traffic it can handle. Adding more mixes extends its capacity, but the drastic imbalance between mixes and system users poses a problem. In addition, traffic analysis attacks are difficult to counter, especially in systems that aim at providing low latency. Like legitimate users, attackers can also easily learn about the mixes the system consists of and try to break it by observ-

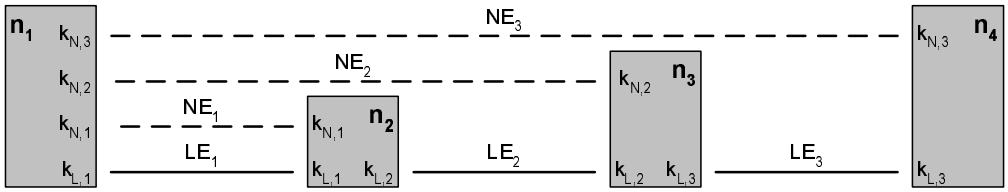



**Figure 1: Layers of encryption.**

ing the traffic at some or all mixes. It is not clear today if this attack can be effectively prevented without employing a constant stream of messages between two mixes. This results in vast amounts of dummy traffic overhead, which seems unacceptable in today's Internet. End-to-end traffic analysis is even more difficult to counter. To protect users from this attack, a constant traffic flow must be established between each user's end-system and the first mix in their anonymous paths. Considering the large number of users one mix handles at the same time, a lot of bandwidth of a mix is absorbed just by the dummy traffic sent to and received from these end-systems. Finally, legal attacks are another major threat. Several governments do not like the idea of anonymity in the Internet. Law enforcement could hinder institutions from operating a mix. Assume a university operates a mix: the government just contacts the key persons at that university and threatens to stop all current and future funding of their research if the service offered by the mix is not discontinued within one week. Because of these limitations, we do not believe that mix networks based on a relatively small set of static mixes are the best way to achieve anonymity in the Internet.

In this paper, we introduce MorphMix, a peer-to-peer based system to enable anonymous Internet usage. We do no longer distinguish between users and mixes. Rather, each user is also a mix at the same time – all participants are equal peers. We believe a peer-to-peer environment eliminates or reduces the drawbacks of the traditional model. However, we also introduce many new challenges. We do no longer have a stable set of highly reliable mixes, but rather a dynamic system of unreliable nodes that may join and leave at any time. Some nodes have good network connectivity, while others employ only slow dial-up connections. We also do not rely on a global public key infrastructure (PKI) providing digital certificates because this will not be realized in the near future and because we want each user be easily able to run a node. This implies that it is not possible to unambiguously authenticate other peers. Finally, allowing anyone to participate makes it is easy for a malicious node (and multiple colluding nodes) to join the system.

With MorphMix, we have several goals in mind: (1) joining the system should be easy for anybody having access to a computer with a public IP address that is connected to the Internet. (2) The bandwidth overhead should be kept low. In particular, we do not want to employ cover traffic. (3) MorphMix should protect from an adversary operating several malicious nodes to break the anonymity of legitimate users. (4) The system should make successful traffic analysis attacks very difficult. (5) MorphMix should be able to efficiently cope with a large number of participating nodes. (6) The end-to-end performance should be acceptable despite the dynamic environment and unreliable nodes.

In the next section, we look at the basic design of MorphMix. Section 3 describes how anonymous paths are established while section 4 discusses how the size and dynamism of the system help to protect from traffic analysis attacks. In section 5, we show how to detect attacks from cooperating malicious nodes. Section 6 discusses related work and section 7 concludes our work and gives an outlook.

## 2. BASIC ARCHITECTURE AND DESIGN

MorphMix consists of an open-ended set of nodes. A node $i$ is identified by its IP address $\mathrm{ip}_i$. In addition, each node has a key-pair consisting of a private key $\mathrm{PrK}_i$ and a public key $\mathrm{PuK}_i$. This key-pair is generated locally when a node runs for the first time.

MorphMix is a circuit-based mix network. To access the Internet anonymously, a user sets up an *anonymous tunnel*, which starts at her own node, via some other nodes. We name the node that is setting up the anonymous tunnel the *initiator*. The last node of the tunnel is called the *final node* and the nodes in-between are the *intermediate nodes*. We also distinguish between *well-behaving nodes*, which are nodes that do not try to break the anonymity of other users and *malicious nodes*, which can collude with other malicious nodes. We make use of layered encryption similar to the approach proposed by Chaum [5]. Figure 1 depicts a fully set up anonymous tunnel from $\mathrm{n}_1$ via $\mathrm{n}_2$, $\mathrm{n}_3$, and $\mathrm{n}_4$.

All messages exchanged between two nodes have the same length. We denote by $\{m\}_k$ the encryption of message $m$ with a key $k$. When $\mathrm{n}_1$ sends a message $m$ through the anonymous tunnel, it encrypts it repeatedly with the symmetric keys corresponding to the *nested encryptions* (NEs), which results in $\{\{\{m\}_{k_{N,3}}\}_{k_{N,2}}\}_{k_{N,1}}$. A header is prepended, which contains an identifier that has local significance on each link between two nodes to route the message along its tunnel. The header also contains a sequence number to counter replay-attacks and a type to distinguish control and data messages.

Before $\mathrm{n}_1$ sends the message to $\mathrm{n}_2$, the header is encrypted according to the *link encryption* (LE) between $\mathrm{n}_1$ and $\mathrm{n}_2$ using the symmetric key $\mathrm{k}_{L,1}$. When $\mathrm{n}_2$ receives the message, it removes the link encryption using $\mathrm{k}_{L,1}$, removes one layer of encryption using $\mathrm{k}_{N,1}$, determines the next hop according to the identifier in the header, sets the fields in the header for the next link, encrypts the header according to the link encryption between $\mathrm{n}_2$ and $\mathrm{n}_3$ using $\mathrm{k}_{L,2}$, and sends it to $\mathrm{n}_3$. This continues until the final node is reached, which relays the data to the server $\mathrm{n}_1$ wants to communicate with. Messages are sent back to $\mathrm{n}_1$ in the same way but in opposite order. This time, each node adds a layer of encryption instead of removing one.

An important design decision is whether the mix network operates on top of the IP layer or on the application layer. In the first case, the system is transparent for end-to-end transport and application protocols. Data is extracted at

the initiator after the IP-layer and transported hop-by-hop through the mix network within UDP diagrams. The end-to-end transport or application protocols are responsible to provide a reliable data stream. In the second case, the user's application usually accesses the mix network in the same way a web browser accesses a web proxy: a TCP-connection is set up to an access program running on the initiator's computer, which in turn handles the communication with the mix network. The data is sent within TCP-connections on each link between two mixes. The system is no longer transparent for the applications, and the access program usually needs to understand the protocol of each application it supports.

In traditional mix networks where each link between two mixes carries the data of several users, UDP is the better choice because with TCP, one lost packet between two mixes stalls every user on that link. Similarly, when cover traffic is used, it is virtually impossible to employ a constant traffic flow between two mixes with TCP. Furthermore, UDP makes sense in an environment where all mixes have similar computing power and network connectivity. Each link between two mixes can be tuned to its maximum throughput without having too many lost datagrams.

In MorphMix, we do not employ cover traffic and due to the large number of mixes, no link is used by very many users at the same time. Furthermore, given the heterogeneity of the nodes, using TCP makes life much easier. With UDP, two nodes would have to employ some sort of flow control between them in order not to lose so many packets that the end-to-end performance would get unacceptable. It is questionable if one could do much better than using TCP directly. A mix network operating on top of the IP-layer also requires that data can be extracted from the protocol stack, i.e. from the kernel space. This is usually not possible without special privileges. Conversely, an application-level mix network operates completely in the user space. We have therefore decided to implement MorphMix as an application-level mix network using TCP between mixes. Although this means losing the transparency of the system to transport and application protocols, we believe it serves the heterogeneity and dynamism of MorphMix better.

# 3. ANONYMOUS TUNNEL SETUP

## 3.1 Selecting the Next Hop

In MorphMix, the initiator selects only the first intermediate node and each node along the anonymous tunnel then picks the following node. This has one big advantage: each node only needs to know about some other nodes. They can communicate with each other and exchange control information to learn which of them have spare resources to accept new anonymous tunnels. Conversely, assume the initiator would select all nodes of an anonymous tunnel. Except for the first intermediate node, it has no idea about the current status of the other nodes, e.g. if they are actually willing to accept further anonymous tunnels. For such a system to work efficiently, a lookup-service would be required. The lookup-service could be queried to get nodes that are currently willing to accept anonymous tunnels. There exist scalable peer-to-peer lookup services such as Chord [20], but the frequent joins and leaves of nodes and the continuously changing state of each node would generate a lot of traffic only to keep the information provided by the lookup-service up-to-date. Letting each node select the next hop makes MorphMix highly scalable because a node only has to manage its local environment. Independent of the system size, a node only cares about a relatively small number of other nodes at any time.

There is one problem with this approach: once we hit a malicious node that wants to collect data about anonymous tunnels, this node could either simulate all remaining hops by itself or use an accomplice as the next hop. We will show in section 3.3 how to solve this.

## 3.2 Local Environment and Peer Discovery

At any time, a node knows about some other nodes, i.e. their IP addresses and public keys. We say that two nodes are *connected* if they have currently established a link encryption. The set of nodes a node $a$ is connected to are $a$'s *neighbors*. Two connected nodes exchange control information, which tells them if the other peer is willing to accept further anonymous tunnels. They can also check the quality of the link by using ping-messages to find out if it actually makes sense to use that link to set up anonymous tunnels. So at any time, a node is connected to some other nodes and knows which of them would currently accept being selected as the next hop in an anonymous tunnel.

There are different ways to learn about other nodes: to join, one must know at least one currently active node. This can be done via a local cache where the node tries contacting nodes that have been active previously, by querying some nodes that are known to be always up, or by contacting some information servers that know about "several" currently active nodes. With several nodes, we mean that such a server knows about a variety of nodes but it does not care about what percentage of all nodes it actually knows. Each participating node contacts some of these servers from time to time and tells them about the nodes it currently knows and gets some other active nodes in return. The servers quickly forget nodes that haven't been advertised in a while and always return a random set of nodes when being queried. This guarantees that a node can learn about a variety of other nodes in a short time.

It is important that different sources are contacted to learn about other nodes. If a newly joining node contacts a single node and that node happens to be part of a large set of colluding malicious nodes, then the joining node would probably only learn from other nodes in that collusion, which again would tell it about other nodes in the collusion and so on. Learning about nodes via different sources should significantly reduce this problem.

## 3.3 Setting up the Link and Nested Encryption

When node $a$ wants to set up the link encryption with another node $b$, it first establishes a TCP-connection with $b$. $a$ then selects a random bit-string that serves as the symmetric key for the link encryption. The key is encrypted with $b$'s public key and sent to $b$.

Setting up a nested encryption takes place between the initiator and a node along the anonymous tunnel. The goal is to establish a symmetric key known only to the two endpoints of the nested encryption. Since the initiator does not know the nodes and their public keys along its tunnel beforehand (except the first intermediate node), we use the Diffie-Hellman (DH) [9] key-exchange. If the initiator simply sent its half of the DH key-exchange to node $b$ responsible

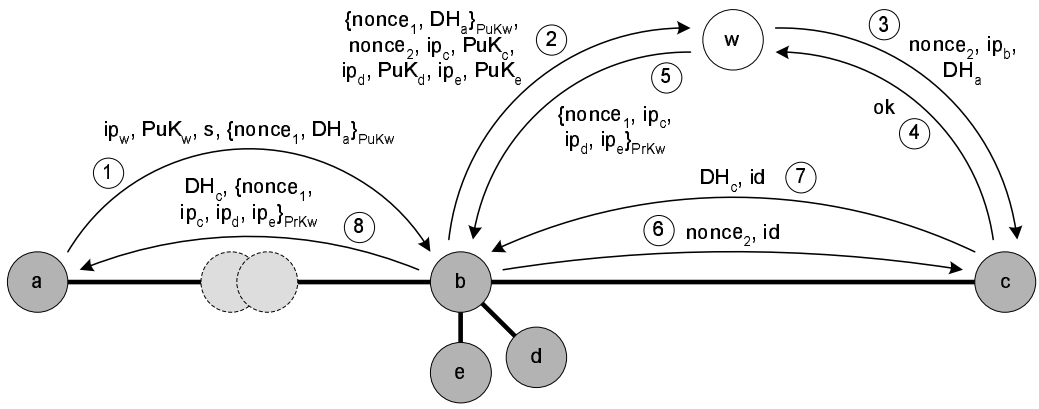



**Figure 2: Setting up the nested encryption**

for selecting the next hop $c$, $b$ could easily play the role of $c$ (and other nodes following $c$) itself without the initiator noticing this. To counter this attack, we must not allow $b$ to see the initiator's half of the DH key-exchange in the clear. To solve this problem, we introduce the notion of a *witness*. For each hop, the initiator selects a witness randomly from the nodes it currently knows. The witness' task is to act as a third party in the process of selecting the next hop of an anonymous tunnel. Figure 2 illustrates the procedure to set up the nested encryption.

Node $a$ is the initiator. We assume the tunnel has already been set up to node $b$ (via zero or more intermediate nodes). In addition, $b$ has currently three connections established to nodes $c$, $d$, and $e$ that are willing to accept further anonymous tunnels. To set up the nested encryption to the next node, the following steps are carried out:

1. $a$ picks a witness $w$ randomly from the set of nodes it currently knows. It generates its half of a DH key-exchange ($\mathrm{DH}_a$) and $\mathrm{nonce}_1$ to prevent replay attacks. $\mathrm{nonce}_1$ and $\mathrm{DH}_a$ are encrypted using $w$'s public key $\mathrm{PuK}_w$, resulting in $\{\mathrm{nonce}_1,\mathrm{DH}_a\}_{PuK_w}$. $a$ also specifies $s$, which is the number of nodes $b$ has to offer to $w$ in message 2. Here, we assume $s = 3$. $a$ then sends a message to $b$ consisting of $w$'s IP address $\mathrm{ip}_w$, $\mathrm{PuK}_w$, $s$, and the encrypted nonce and DH parameters. The message tells $b$ to append a node to the tunnel using witness $w$.
2. $b$ receives the message and sets up a link encryption to $w$, using $\mathrm{ip}_w$ and $\mathrm{PuK}_w$. It generates $\mathrm{nonce}_2$, which is used to recognize message 6. $b$ generates a message containing the encrypted nonce and DH parameters from $a$, $\mathrm{nonce}_2$, the IP addresses of 3 potential next hop nodes ($\mathrm{ip}_c$, $\mathrm{ip}_d$, and $\mathrm{ip}_e$) and their public keys ($\mathrm{PuK}_c$, $\mathrm{PuK}_d$, and $\mathrm{PuK}_e$) and sends it to $w$. We name the list of IP addresses offered by $b$ the *selection* of $b$.
3. $w$ receives the message and randomly picks one node from the selection of $b$ as the next hop. In figure 2, $w$ picks node $c$ and establishes a link encryption with $c$ using $\mathrm{PuK}_c$. $w$ also decrypts $\mathrm{nonce}_1$ and $\mathrm{DH}_a$ using its private key $\mathrm{PrK}_w$, generates a message consisting of $\mathrm{nonce}_2$, $\mathrm{ip}_b$, and $\mathrm{DH}_a$, and sends it to $c$.
4. $c$ gets the message and checks if it is indeed willing to accept an anonymous tunnel from $b$. If yes, $c$ generates an ok-message and sends it back to $w$.
5. $w$ receives the ok-message and generates a receipt for $a$. The receipt contains the IP addresses offered by $b$ and is signed by $w$ using $\mathrm{PrK}_w$. The first IP address in the receipt is the one $w$ has picked as the next hop. The receipt also contains $\mathrm{nonce}_1$ to guarantee its freshness and is sent to $b$.
6. $b$ receives the message from $w$ and learns that $w$ has selected $c$ as the next hop. It generates a message containing $\mathrm{nonce}_2$ and the identifier id to be used to identify data belonging to this anonymous tunnel on the link between $b$ and $c$. After having sent this message, $w$'s task is completed and the connection between $b$ and $w$ can be torn down again.
7. $c$ gets the message and sends its part of the DH key-exchange ($\mathrm{DH}_c$) back to $b$ via a message identified with id.
8. $b$ generates a message consisting of $\mathrm{DH}_c$ and the receipt from $w$ and sends it to $a$.

If anything fails, a nok-message is sent back to $a$ and $a$ can either decide to tear down the tunnel completely or try again. Note that the same procedure as above is used to add the hop directly following the initiator. Of course, $a$ could simply pick that node by itself and directly establish the nested encryption. However, this would tell the node following $a$ that $a$ is the initiator of the anonymous tunnel.

Before analyzing the attacks, we identify the two main features of the nested encryption setup. The first is making sure that $b$ does not learn $a$'s half of the DH key-exchange as this would easily enable $b$ to simulate all remaining hops by itself. This is achieved by encrypting $\mathrm{DH}_a$ for $w$ and only sending it in the clear from $w$ to $c$. $b$ never sees $\mathrm{DH}_a$ in non-encrypted form. The second is preventing $b$ from selecting the next hop purely by itself. This is achieved by having $b$ offering a selection of possible next hops to $w$ and $w$ selecting one of them. This guarantees that $b$ cannot predict which of the nodes in the selection is going to be picked as the next hop and makes it much more complicated for $b$ to determine the next hop. In particular, if $b$ wants to make sure that $c$ is in the same set of colluding nodes as itself, then all nodes in the selection of $b$ must be in that collusion.

## 3.4 Analysis of the Nested Encryption Setup

We only briefly discuss the most important attacks. For a more detailed analysis, refer to the technical report [16].

If $b$ wants to simulate the next hop $c$, it can provide $w$ in message 2 with fake public keys it knows the private keys

of, intercept message 3 and act as $c$ itself. To do so, $b$ needs active control over the link between $w$ and $c$ to intercept and inject data packets. However, $b$ cannot predict which witness $a$ is going to choose, so $b$ cannot prepare itself in advance and it is difficult to intercept packets close to $w$. It seems more realistic for $b$ to intercept packets close to $c$, especially as it is $b$ that selects the list of nodes in message 2. To make this attack as difficult as possible, we require that all IP addresses offered by $b$ in its selection and $b$'s own IP address must not have similar IP prefixes. We will discuss the number of IP addresses $b$ has to offer $c$ in section 5.

If $b$ and $w$ are in the same set of colluding nodes, it is trivial for $b$ to simulate the next hop $c$ because $w$ can provide $\mathrm{DH}_a$. Additionally, $w$ can generate a receipt at will in message 5. However, since $a$ chooses randomly a different witness for each hop, the probability that all witnesses are cooperating with $b$ is quite small if we assume that only a relatively small portion of all nodes is malicious. As soon as the witness for a link is not cooperating with $b$, it gets much more difficult again for $b$ to simulate the next hop.

If we assume $b$ is part of a larger set of cooperating malicious nodes, then $b$ simply lists a subset of these malicious nodes in message 2 and it is guaranteed that the next hop is also part of the cooperating set. As we require that the IP addresses must not have similar IP prefixes, the malicious nodes must reside in different subnets, which complicates the attack. Nevertheless, if an adversary manages to accumulate several nodes located in different areas of the Internet, then this attack is quite easy to carry out.

We conclude that the most realistic attack is the one where a set of cooperating malicious nodes tries to control as many nodes along an anonymous tunnel as possible by offering many or exclusively nodes from their collusion in their selections. All other attacks require active control over several network links and are therefore much harder to carry out. In addition, if something like a world-wide PKI got deployed, the use of digital certificates would defeat those attacks impersonating another party would no longer be possible if $c$ signed message 7.

One final note regarding the selection of a witness to add a hop to an anonymous tunnel. Since the witness knows its neighbors, the initiator should select a witness from the set of nodes it knows but never from those it is currently connected to. This is also true when setting up the nested encryption with the first intermediate node, where the initiator $a$ contacts the witness directly: if the witness were always chosen from the current neighbors, the witness could conclude with high probability that $a$ is indeed the initiator of the anonymous tunnel.

## 4. TRAFFIC ANALYSIS ATTACKS

In this section, we look at how the large number of mixes and the dynamism of MorphMix helps to protect from passive traffic analysis attacks.

If a global eavesdropper can observe every single MorphMix node, we are doomed. Due to the limited mix functionality of the nodes – in particular because we choose not to employ cover traffic – such an adversary should be able to break the anonymity of all MorphMix users by means of timing attacks at the nodes along anonymous tunnels or end-to-end timing attacks at the first and final nodes. The question is if such an attacker is a reasonable assumption. As mentioned in section 1, traditional, static mix networks are composed of a small number of well-known mixes. This implies that only a few Internet service providers (ISPs) have to combine data to get a complete log of all data flowing through the mix network. Although the community has been arguing for years if the threat model with a global eavesdropper is realistic, a lot of effort has been spent to harden mix-based systems (forward-only and circuit based) and to find attacks on them [1, 3, 7, 8, 11, 18, 19, 21, 23]. The conclusion is that – at least for circuit-based systems – a high level of anonymity against a global observer can probably not be achieved without employing vast amounts of dummy traffic. Even that may not be enough to stop a global active attacker capable of randomly blocking links in the system. To successfully resist such an attacker, the whole system needs to be stalled in case the data flow along any link between two mixes stops, which probably renders such a system not very useful in practice. Designing a system that provides perfect anonymity against a global active attacker while giving its user's satisfactory end-to-end performance for near-real-time applications is very difficult and maybe not possible.

The difference in MorphMix is that because of the large number of mixes, a global observer seems extremely unlikely. The data of very many ISPs around the world have to be combined to get the whole picture. In addition, due to the large number of active nodes in the system, there exists a huge number of potential paths a message can take as it travels through the network. Nodes appear and disappear and the whole system is dynamic and changes continuously, which makes it virtually impossible for anyone to get knowledge of the whole network at any time.

To take full advantage of the large number of nodes, it is not enough for a node to discover some peers once it has become active and to communicate with them for hours. The reason is that this would greatly limit the possible previous and next hops of anonymous tunnels through a node during the time it is active. Rather, each node should constantly try to learn about other peers that can be used as possible next hop nodes in anonymous tunnels and forget about those it has been using for a while. As a result, each node can potentially be connected to any other node at a time, which implies that anonymous tunnels can follow any possible path through the network. Similarly, when acting as the initiator of anonymous tunnels, a node does not establish one tunnel and use it for a while, but keeps setting them up in the background. The goal is to have some anonymous tunnels established at any time. Each tunnel is only used for a relatively short time and several can be used in parallel, if the application makes use of multiple end-to-end connections at a time (think of communicating with a web server and receiving the embedded objects within a page through various TCP-connections). Changing anonymous tunnels frequently is also beneficial for the collusion detection mechanism (see section 5), and having more than one anonymous tunnel available at any time helps coping with unreliable nodes or nodes that offer poor performance at times: if the throughput of an anonymous tunnel is very bad or it has stopped working completely because an intermediate node has gone down, the tunnel is simply dropped and another one is used.

Because of the size and dynamism of MorphMix, it is unlikely an attacker can systematically observe a particular user by monitoring all mixes along his anonymous tunnels.

Similarly, end-to-end traffic analysis attacks are difficult to carry out because there are so many possible exits for each anonymous tunnel and because there are no longer so easily identifiable entry-points (the link between the user's computer and the first mix) into the system as in the traditional model. We also argue that using cover traffic would not add much more to the resistance of MorphMix. In particular, keeping up constant traffic flows between nodes in a way that really protects from traffic analysis attacks without significantly degrading the end-to-end performance would be very difficult in a highly dynamic environment with unreliable nodes.

We conclude that a limited eavesdropper that is able to monitor several nodes but not a significant portion of the system may occasionally break the anonymity of a user if he manages to observe at least the traffic at the initiator and final node of an anonymous tunnel. As soon as the user switches to another tunnel, her identity is protected again. This implies that MorphMix is well suited to protect its users from long-term profiling without guaranteeing the anonymity of every single transaction.

## 5. DETECTING COLLUSION ATTACKS

It is a hard problem to detect nodes that are just collecting data but otherwise offer good service. However, there is one key difference between an anonymous tunnel that was set up via well-behaving nodes and one that is partly composed of cooperating malicious nodes: in the first case, each node is selected more or less randomly among all active nodes in the system, while in the second case, nodes from the malicious set appear with higher probability. Detecting nodes that appear more often together in anonymous tunnels than others can only work when a user has set up and used a variety of different anonymous tunnels, which is another argument to support frequently changing the tunnels one is using.

In this paper, we describe the basic collusion detection mechanism. It does not yet take prefixes of IP addresses into account: two IP addresses are equal if they match in every bit, otherwise they are completely different. The collusion detection is based on the receipts a user gets from different witnesses during the setup of anonymous tunnels (figure 2 messages 5 and 8). A receipt contains the possible next hops offered to the witness (figure 2 message 2). The first node in a receipt is the one selected by the witness, which implies the initiator knows which node has offered which selection for each intermediate node in an anonymous tunnel.

Each node maintains an *internal table* that contains a row for each selection it has received during the setup of anonymous tunnels. Each row is a combination of a selection and the node that offered the selection, which we name *extended selection*. If node $b$ has offered the selection $\{\mathrm{ip}_c, \mathrm{ip}_d, \mathrm{ip}_e\}$, the resulting extended selection is $\{\mathrm{ip}_b, \mathrm{ip}_c, \mathrm{ip}_d, \mathrm{ip}_e\}$,

We now describe the computations a node performs to determine if an anonymous tunnel is composed of colluding nodes or not. For each new extended selection, a node computes the *correlation* according to algorithm 1:

ALGORITHM 1. *Computing the correlation of an extended selection*

1. *Build a set $ES_N$ consisting of the nodes of the new extended selection.*
2. *Define a result set $ES_R$ which is empty at first.*
3. *Compare each extended selection $ES_T$ in the internal table with $ES_N$. If $ES_N$ and $ES_T$ have at least one element in common, add the elements of $ES_T$ to $ES_R$.*
4. *Count each occurrence of elements in $ES_R$ that appear more than once and store the result in $m$.*
5. *Count the number of elements that appear only once in $ES_R$ and store the result in $u$.*
6. *Compute the correlation $c$ which is defined as $c = m/u$ if $u > 0$, or $\infty$ otherwise.*

We argue that the correlation is in general relatively big if the new extended selection contains many or only colluding nodes. Colluding nodes (1) select other colluding nodes with high probability and (2) are selected by other colluding nodes with high probability. This follows from our assumption we stated in section 3.4 where we said that attacks by a cooperating malicious set of nodes are most likely. Similarly, well-behaving nodes (3) pick nodes for the selections they offer from the set of all other nodes and (4) are picked by all other well-behaving nodes. In step 3 of algorithm 1, we want to find out what the nodes in the new extended selection have done before, i.e. in what extended selections they have appeared before and collect all extended selections in the internal table that contain elements of the new extended selection in a set $\mathrm{ES}_R$. For reasons (1–4), we can state the following properties about the set $\mathrm{ES}_R$:

1. If $\mathrm{ES}_N$ mainly consists of colluding nodes, $\mathrm{ES}_R$ will contain relatively few different nodes and many occurrences of several colluding nodes. This implies a big $m$ and a small $u$, resulting in a big $c$.
2. If $\mathrm{ES}_N$ mainly consists of well-behaving nodes, $\mathrm{ES}_R$ will contain relatively many different nodes with only a few of them occurring several times. This implies a small $m$ and a big $u$, resulting in a small $c$.

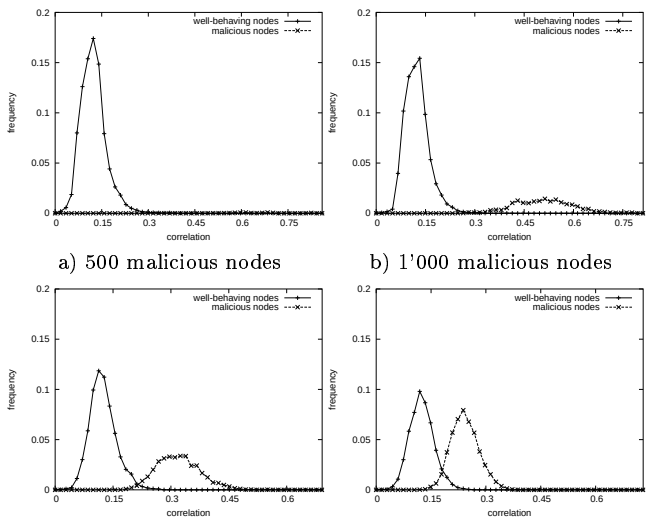
Why do we not simply count how many times the elements in $\mathrm{ES}_N$ show up in the internal table? This would work if we assumed that every node in the system was selected by well-behaving nodes with the same probability. In this case, colluding nodes would stand out since overall, they would be selected more often than the well-behaving ones due to their preference in the selections of colluding nodes. However, in a real-world scenario, some nodes will be much more popular than others because of their spare bandwidth and computing power. Counting only the number of occurrences of nodes in the internal table, one could wrongly suspect the very popular nodes to build a colluding set, which would greatly hurt the performance of the whole system. What distinguishes well-behaving popular nodes from colluding nodes is that although the popular nodes appear frequently in selections of well-behaving nodes, less popular nodes appear in the same selections, too. Consequently, the variety of nodes being selected by well-behaving nodes is always bigger than the one selected by malicious nodes, even if there are some very popular nodes. Similarly, it would not be sufficient to look only at $m$ instead of the ratio $m/u$. With several popular well-behaving nodes in an extended selection, $m$ can get quite big and the nodes in the extended selection could again be suspected to build a colluding set. This is why we take $u$ into account: $u$ tends to get relatively big compared to $m$ when the new extended selection contains mainly well-behaving nodes – independently of the popularities of the nodes, but is relatively small compared to $m$ when the extended selection consists of several malicious nodes.

## 5.1 Detecting Malicious Tunnels

We have argued that high correlations are an indication for colluding nodes. However, we have not given a limit above which extended selections get suspicious. The problem is that there is no such fixed limit. The correlations depend on the number of nodes in the system, their popularities, the number of nodes in a selection, and the size of the internal table.

A node remembers the correlations it has computed over time and represents them as a distribution function. It is implemented as an array, whereas each slot of the array corresponds to a particular discrete correlation. If a new correlation $c$ is computed, it basically affects the slot closest to $c$ by incrementing its value by 1. However, in order not to let grow the values in the array indefinitely, they follow an exponential weighted moving average (EWMA) with parameter $\alpha$. $\alpha$ is slightly smaller than 1 and depends on the number of extended selections in the internal table. After a new correlation has been computed, the value in each slot is multiplied with $\alpha$, and $(1-\alpha)$ is added to the slot that corresponds to the new correlation.

We analyze how the correlation distribution looks. We assume a system with 10'000 nodes, where some of them are malicious and in the same colluding set. Each node is equally popular. We set up 5'000 anonymous tunnels, whereas each tunnel consists of 5 nodes in total. This means that the initiator gets 3 different selections during the setup of each tunnel, one from each of the intermediate nodes. Each selection contains 10 nodes, which is a reasonable selection size in a system with 10'000 nodes (see section 5.3). For now, we assume that malicious nodes offer only other malicious nodes from their collusion in their selections, i.e. selections from malicious nodes contain 10 malicious nodes. Figure 3 shows the correlation distribution when 5, 10, 20, or 30% of all nodes are malicious.




a) 500 malicious nodes b) 1'000 malicious nodes

c) 2'000 malicious nodes d) 3'000 malicious nodes

**Figure 3: Correlation distribution with 10'000 nodes**

We can see the contributions of well-behaving and malicious nodes to the correlation distribution. In general, it results in two peaks, one on the left from the well-behaving nodes and one on the right from the malicious nodes. The more malicious nodes there are in the system, the bigger the right peak gets and the closer the two peaks move together.

The strategy a node follows to detect malicious anonymous tunnels is as follows: At any time, the node knows the correlation distribution it has generated based on selections it received previously. Based on this distribution, the node determines a *correlation limit*, which should have the property that if the correlation of a new extended selection is smaller than this limit, then the node that offered the corresponding selection is well-behaving with a high probability. Similarly, the extended selection corresponding to the selection of a malicious node should yield a correlation that is above the limit with high probability. If the correlations of all extended selections of an anonymous tunnel are below that limit, then the anonymous tunnel is considered *good*. If only the final node in the tunnel is malicious, then this is difficult to detect because it does not offer a selection. However, this final node cannot learn anything about the anonymous tunnel by itself. But if the correlation of at least one extended selection is above the limit, the tunnel is considered *malicious* and will not be used. The difficulty of determining this limit is that the node only knows the correlation distribution of all nodes, i.e. the sum of the contributions of well-behaving and malicious nodes in figure 3.

The steps the initiator carries out during the setup of an anonymous tunnel to determine whether it is considered good or malicious are listed in algorithm 2:

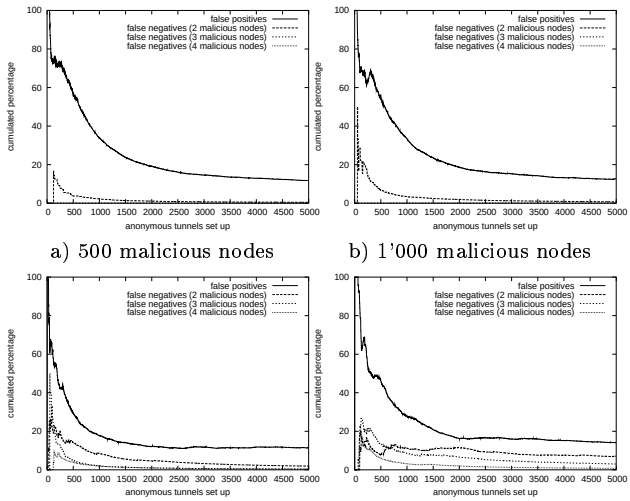
ALGORITHM 2. *Determining if an anonymous tunnel is good or malicious*

1. *Initialize a variable* `rejectTunnel` *to false.*
2. *Get the next extended selection $ES_N$ of the anonymous tunnel.*
3. *Compute the correlation $c$ of $ES_N$.*
4. *Determine the limit correlation $c_l$ from the correlation distribution.*
5. *If $c$ is greater than $c_l$, set* `rejectTunnel` *to true.*
6. *Add $c$ to the correlation distribution and add $ES_N$ to the internal table.*
7. *If there are more intermediate nodes following in the tunnel, go to step 2.*
8. *If* `rejectTunnel` *is true, reject the tunnel. Otherwise it is considered good.*

We now analyze how well our algorithm performs. There are two figures we are evaluating: *false positives*, i.e. the number of good anonymous tunnels that were wrongly classified as malicious, and the *false negatives*, which are those anonymous tunnels that have been classified as good but actually contain more than one malicious node. According to our assumption that malicious nodes present only other malicious nodes in their selections, it is guaranteed that the $m$ malicious nodes in a tunnel are always the last $m$ hops of that tunnel. A tunnel consisting of $n$ nodes may contain $1 \ldots (n-1)$ malicious nodes. Tunnels where only the final node is malicious cannot be detected but do not pose a problem, as mentioned above. Consequently, we try to detect tunnels consisting of $2 \ldots (n-1)$ colluding nodes. We therefore split the false negatives further depending on the number of malicious nodes anonymous tunnels contain. If a tunnel contains 5 nodes, then there are false negatives with 2, 3, or 4 malicious nodes.

Figure 4 shows the false positives and negatives for the setting in figure 3. The graphs show the cumulated per-

centages of false positives and negatives after $n$ anonymous tunnels have been set up. For instance, in figure 3a, the line with the false positives shows about 20% false positives after 2'000 anonymous tunnels. This means that 20% of all good anonymous tunnels were wrongly classified as malicious during the setup of the first 2'000 anonymous tunnels. The table lists the absolute figures of false positives and negatives after all 5'000 anonymous tunnels have been set up.




a) 500 malicious nodes b) 1'000 malicious nodes

c) 2'000 malicious nodes d) 3'000 malicious nodes

| | 5% malicious | 10% malicious |
|---|---|---|
| f. pos. | 505 of 4291 → 11.77% | 450 of 3637 → 12.62% |
| f. neg. (2) | 1 of 224 → 0.45% | 3 of 411 → 0.73% |
| f. neg. (3) | 0 of 244 → 0.00% | 0 of 449 → 0.00% |
| f. neg. (4) | 0 of 241 → 0.00% | 0 of 503 → 0.00% |
| | 20% malicious | 30% malicious |
| f. pos. | 292 of 2566 → 11.38% | 244 of 1733 → 14.08% |
| f. neg. (2) | 13 of 681 → 1.91% | 52 of 736 → 7.07% |
| f. neg. (3) | 3 of 789 → 0.38% | 33 of 1100 → 3.00% |
| f. neg. (4) | 4 of 964 → 0.41% | 11 of 1431 → 0.77% |

**Figure 4: False negatives and positives with 10'000 nodes**

We see that false negatives mainly occur when only a few tunnels have been set up. The reason is that in the beginning, it is difficult to determine if a new extended selection is good or malicious because the internal table of the initiator does not yet contain enough extended selections. After this initial phase, however, only very few malicious tunnels remained undetected. False positives happen from time to time, which is caused by the fact that the correlation limit is always chosen to minimize the false negatives at the cost of a few false positives. We can also see that with more malicious nodes, it takes longer until we can detect false negatives with high probability, which makes sense because more anonymous tunnels are needed to learn enough about the adversary. This is confirmed by looking at the number of completely compromised tunnels (those consisting of 4 malicious nodes) in the table in figure 4: with 5 or 10% malicious nodes, we missed none of them, with 20% we missed 4, and with 30% malicious nodes we missed 11 fully compromised tunnels until the initiator had collected enough information.

The collusion detection mechanism has its limit. If the amount of malicious nodes is increased to 50% and beyond, detecting malicious tunnels is no longer possible because the two peaks in the correlation distribution merge into one. Nevertheless, we conclude our mechanism to detect malicious tunnels basically works very well. Of course, there is a learning phase, but once the initiator has accumulated enough information, virtually all malicious tunnels are detected. However, it should be noted that our measurements are based on the assumption that well-behaving and malicious nodes are equally popular and that malicious nodes offer only other malicious nodes from the same collusion in their selections. We will examine different adversarial games in section 5.2.

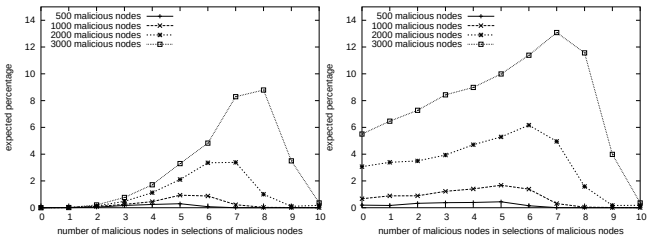
The fact that it takes setting up some anonymous tunnels until a node can make reasonable judgments about whether a tunnel is good or malicious has some implications. First of all, to not lose the knowledge about previously established tunnels in case a node has been inactive for a while, its full internal table is periodically stored on disk. But besides that, MorphMix provides incentive for a user to keep her node active even when she does not need to access the Internet anonymously: the node continues to set up anonymous tunnels to collect information about the system, which increases the user's protection from collusion attacks, and at the same time this adds to the system's size and dynamism to increase its resistance to traffic analysis attacks.

## 5.2 A More Clever Adversary

We have seen in the previous section that life gets difficult for the adversary if the nodes he controls offer other malicious nodes from the same collusion too aggressively. A different adversarial game is to offer not only malicious nodes but also well-behaving nodes in their selections. According to algorithm 1, this should bring the peaks resulting from the selections of well-behaving and malicious nodes closer together and make it more difficult for the initiator to detect compromised tunnels.

We use the same basic setting as in section 5.1 and vary the number of malicious nodes in selections of malicious nodes from $0 \ldots 10$. In contrast to section 5.1, it is now no longer the case that all remaining nodes of a tunnel are malicious once a malicious node has been hit because a witness can choose a well-behaving node from the selection of a malicious node. Consequently, it is now possible that the adversary controls the first intermediate and the final node of an anonymous tunnel, but not necessarily all others in-between. As we do not employ cover traffic, it could be the case that an advanced adversary makes use of timing attacks to learn these two nodes belong to the same tunnel, which means he would have fully compromised the tunnel. We therefore look more closely at two cases: (1) the adversary controls all nodes following the initiator along an anonymous tunnel and (2) the adversary controls at least the first intermediate and the final node. Figure 5 shows the percentage of all anonymous tunnels the adversary is expected to compromise according to the two cases described above.

We see that the adversary's chances to fully compromise anonymous tunnels increases compared to figure 4. For instance, figure 5a shows that an adversary controlling 1'000 nodes is likely to fully compromise nearly 1% of the anonymous tunnels if the nodes he controls offer 5 malicious nodes in their selections. However, this is still significantly better




a) adversary controls all nodes following the initiator

b) adversary controls first intermediate and final node

**Figure 5: Expected percentage of compromised tunnels with a less aggressive adversary**

than the expected 10% fully compromised tunnels without using the collusion detection mechanism. Figure 5a also shows that if the number of nodes in the collusion increases, then malicious nodes can offer more malicious nodes in their selections without being detected by the initiator. Controlling 3'000 nodes and offering 8 malicious nodes in the selections allows the adversary to fully compromise nearly 9% of all tunnels. Without employing a collusion detection mechanism, his probability of success would be 30%. Note that it is possible to further reduce the percentages in figure 5a by using more intermediate nodes in an anonymous tunnel, but at the price of an increased end-to-end delay. For instance, using 7 instead of 5 nodes reduces the maximum expected percentage of fully compromised tunnels to about 0.15% with 1'000 and below 1% with 2'000 malicious nodes.

Figure 5b shows that an adversary controlling 1'000 nodes can expect to control the first intermediate and final node in 1.7% of all anonymous tunnels if malicious nodes offer 5 other malicious nodes in their selections. This is slightly above the 1% he would control if he played fair, i.e. if he offered well-behaving and malicious nodes in the same way as well-behaving nodes did. With 3'000 malicious nodes, this goes up to about 13% compared to 9% if the malicious nodes played fair. Using more nodes in an anonymous tunnel again brings down the percentages.

Although the adversary is able to fully compromise a few anonymous tunnels using the less aggressive strategy discussed above (either trivially by controlling all nodes following the initiator or by controlling at least the first intermediate and the final node and making use of timing attacks), his abilities are still very limited. First of all, he does not know if the first node he controls is really the first intermediate node, which implies he cannot know for sure who the initiator is. Second, although he can expect to compromise some anonymous tunnels, he cannot mount a targeted attack on a node to compromise all its tunnels where he controls the first intermediate node during the next hour or so. He can continuously try, but only occasionally he will manage to control enough nodes along a tunnel to fully compromise it without being detected by the initiator.

Another strategy of the adversary could be to make sure the nodes he controls are not very popular. This could be achieved by telling their neighbors that they are not willing to accept further anonymous tunnels. The main idea behind this strategy is to have only a few extended selections from malicious nodes in the internal tables of the initiators to keep their correlations small, which should bring the peaks resulting from the selections of well-behaving and malicious nodes closer together.

Figure 6 shows the adversary's expected percentage of compromised tunnels if he varies the relative popularities of the malicious nodes from 0.05 . . . 1.0 of the well-behaving nodes.

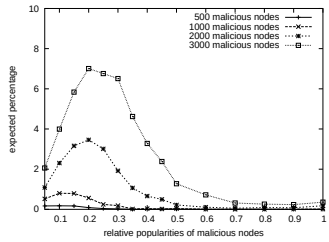



**Figure 6: Expected percentage of compromised tunnels with less popular malicious nodes**

Here again, the adversary's changes increase compared to figure 4. When controlling 1'000 nodes, he manages to compromise nearly 1% of all tunnels when keeping the malicious nodes' relative popularities at 10–15%. With 3'000 malicious nodes, it increases to 7% when keeping the malicious nodes' relative popularity at 20%. It is again possible to further reduce the percentages in figure 6 by using more intermediate nodes in an anonymous tunnel, but at the price of an increased end-to-end delay. Using 7 instead of 5 nodes reduces the maximum expected percentage of fully compromised tunnels to about 0.3% with 1'000 and 0.55% with 2'000 malicious nodes. We conclude the collusion detection mechanism still works well also against this adversarial game. In particular, an adversary controlling not more than about 10–20% of all nodes will not be able to fully compromise more than very few anonymous tunnels.

In general, the adversary remains undetected as long as he makes sure the density of extended selections containing many malicious nodes does not grow beyond a certain threshold an initiator's internal table. This means he can send some malicious external selections to an initiator during the setup of anonymous tunnels until this threshold is reached without attracting attention. Extended selections do not remain in the internal table forever (see section 5.3), so once the threshold is reached, the adversary has to wait until the initiator has forgotten about some of these malicious extended selections before he can successfully attack again. If we assume a user has a "clean" internal table in the sense that it does not contain malicious extended selections, the adversary can either attack her aggressively for a short time to compromise relatively many tunnels and then wait a long time until the initiator has forgotten the malicious extensions, or he can choose to "spend his credit" over a longer time to occasionally break an anonymous tunnel.

Note we have deliberately not included the percentages of false positives in figures 5 and 6. For the sake of completeness, there are always about 10% false positives. Additionally, it is always possible to further reduce the rate of false negatives by determining the correlation limit more conservatively, but at the cost of more false positives.

We conclude that MorphMix is well suited to protect its users from long-term profiling attacks carried out by an adversary controlling a limited number of nodes. This result is very similar to our discussion about the possibilities of a limited eavesdropper (see section 4).

## 5.3 Scalability

We have analyzed the influence of several parameters in our system on its behavior and performance [16] and briefly summarize the most important results.

Most parameters depend of the number of nodes in the system. A node remembers all other nodes it has seen as part of selections in a *least recently seen nodes* list. Each entry contains a timestamp that specifies when the node has been seen for the last time. Nodes not showing up for a while are removed from the list. Upon joining the system for the first time, a node has no idea how many other nodes there are and only learns about this after having set up several tunnels. However, observing how fast the correlation distribution starts getting its typical shape allows the initiator to guess the number of nodes in the system.

The first parameter we look at is the size of the selection. In general, larger selections yield better separations of the two peaks in the correlation distribution. However, very large selections require each node to be connected to very many other nodes at one time. We have carried out several measurements and derived a formula that provides a good compromise. If $n$ is the number of different nodes in the system, then the selection size $s$ should be chosen as $s = \max(\lceil 5 \cdot \log_{10} n - 10 \rceil, 1)$ [16]. This means the selection size grows logarithmically with the system size. As an example, with 10'000 nodes in the system, $s$ should be set to 10, as we have done in the examples in sections 5.1 and 5.2.

Another issue is the size of the internal table. The complexity to compute the correlation of a new extended selection is proportional to the number of extended selections in the internal table. We therefore try to keep its size as small as possible to minimize the overhead. The idea is to "forget" old extended selections and to keep only the $k$ *least recently received extended selections* in the internal table. This is not only reasonable to keep the complexity low, but also makes sense because new extended selections give the most accurate picture of the current situation of the system. Like above, we have derived a formula that provides good results. If $\overline{s}$ is the average number of elements in a selection and $n$ the number of nodes in the system, the number of extended selections $k$ in the internal table should be $k = 2 \cdot n/\overline{s}$ [16], which means the internal table grows linearly with the size of the system. Following the example above with 10'000 nodes and a selection size of 10, $k$ would be 2'000.

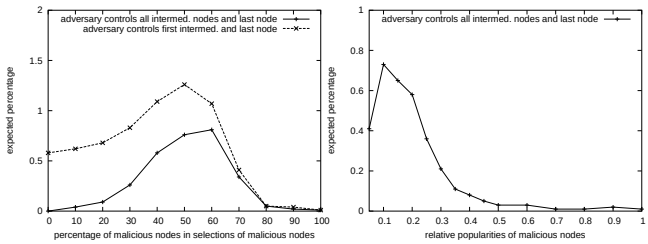
This linear dependency of the complexity to process a new extended selection poses a problem if the system gets very large. We have performed our measurements on a system with a 1 GHz AMD Athlon CPU and 256 MB RAM, running Linux with a 2.4.17 kernel. The software is written in Java and we use Sun's Java 2 SDK 1.4. With $n$ nodes in the system, it takes about $n/2'500$ ms to completely process a new extended selection. With 100'000 nodes, this results in 40 ms, which is acceptable, but with 1'000'000 nodes, this grows to 400 ms, which is no longer insignificant.

## 5.4 A Realistic Scenario

We look at a what we believe is a realistic scenario. There are 100'000 nodes in the system. The popularities of the well-behaving nodes follow a negative exponential distribution where the most popular nodes are 50 times as popular as the least popular ones. We also take into account that nodes enter and leave the system, so at any time not all of the well-behaving nodes are active. In general, we can assume that the popularity and availability of a node are not independent because nodes that are available most of the time often have better network connectivity than those with slow dial-up connections that are online for only an hour or so every day. Nevertheless, it may also happen that some popular nodes are only available every now and then and that some unpopular nodes are nearly always active. We model this by assigning each node an availability between 0 and 1. Popular nodes have generally a higher availability, but there are exceptions. We choose an average availability of 0.25 for the well-behaving nodes, which implies there are about 25% of them active at any time.

To be most effective, the adversary makes sure that as many of the nodes he controls are active. We model this by assigning each malicious node an availability of 0.8. In addition, we use another result from our experiments [16]: the adversary's chances to compromise anonymous tunnels without being detected increase if he manages to keep the popularities of the nodes he controls more or less equal. Therefore, we model the popularities of the malicious nodes also with a negative exponential distribution, but here the most popular nodes are only 5 times as popular as the least popular ones. We also take malicious witnesses into account: if it happens that the witness and the node setting up the next hop are in the same collusion, then the witness generates a fake selection that does not contain any node from the collusion to confuse the initiator.

The initiator sets up 20'000 tunnels. After every 100 anonymous tunnels, the set of active nodes is determined according to their availabilities: a node with availability 0.5 has a 50% probability of being active during the time the next 100 anonymous tunnels are set up. The adversary controls 2'500 nodes. We carry out the same measurements as in section 5.2. Figure 7 depicts the results.




a) less aggressive adversary b) less popular malicious nodes

**Figure 7: A realistic scenario**

We see that the mechanism to detect malicious tunnels copes very well with this dynamic scenario. The results are comparable with the measurements with 10% malicious nodes in figures 5 and 6. This is not surprising because in the dynamic scenario evaluated in figure 7, approximately 25% of the well-behaving nodes and 80% of the malicious nodes are active at any time, which means there are also about 10% malicious nodes among the active nodes at any time. The percentage of false positives is again about 10%.

# 6. RELATED WORK

In section 1, we have already mentioned some systems following Chaum's traditional approach. Here we have a look at two systems based on peer-to-peer technology, Crowds and Tarzan.

Crowds [15] collects users in a group (the "crowd") to browse the Web anonymously. To join, a user contacts a central server and learns about the other members. A user that wants to request a web page forwards the request randomly to another member in the crowd. When a crowd member receives a request from another member, it makes a random choice to either forward the request to another crowd member or submit it to the server the request is intended for. The reply from the server uses the same path back. To the outside, the system provides anonymity in the sense that any crowd member could have requested the web page. Crowds does not make use of layered encryption but uses a shared key that is known to all members in a crowd to link-encrypt the messages. Crowds is similar to our system in the sense that it also implements a "every node is a mix" policy, but does not employ a collusion detection mechanism to protect from collaborating members.

Tarzan [11] is a recent effort to provide a peer-to-peer anonymizing network layer. Tarzan provides anonymous best-effort IP service and is transparent to applications. The system makes use of layered encryption, fixed-length messages, and cover traffic to guarantee high protection against traffic analysis attacks. The cover traffic mechanism is especially worth mentioning: each node maintains a bidirectional packet stream with a fixed number of other nodes (its *mimics*). Anonymous tunnels through a node are only relayed via the node's mimics, which implies that real data are always hidden in the packet streams between the node and its mimics. While this approach limits the possible paths that can be selected for a tunnel, it has the advantage that cover traffic is exchanged only between a few of all potential pairs of nodes. In general, Tarzan has strong anonymity properties. To achieve them, a node cannot simply select its mimics as it likes. Rather, they are selected in a pseudo-random, but universally verifiable way from the pool of all present nodes. Consequently, the probability that a malicious node has only other malicious nodes as its mimics is very small, which implies it is difficult for an adversary to control all nodes in a tunnel. To select the own and verify another node's mimics, a node needs to know about all nodes in the system. Additionally, a node validates each other node upon learning from its presence by contacting it. It is reasonable to assume that Tarzan works quite well even with very many nodes in the system if the participating nodes do not change too frequently. On the other hand, especially the requirement to know about all other nodes leaves open the question how well Tarzan can cope with a dynamic environment where nodes come and go.

Although not directly comparable with our work, there has been another proposal to use witnesses in mix networks [10]. In contrast to our system where witnesses are used to select the next hop randomly, their witnesses are used to discover bad nodes that fail to forward messages to increase the reliability of a mix network.

## 7. CONCLUSIONS AND FUTURE WORK

We have presented MorphMix, a system that enables peer-to-peer based anonymous Internet usage. Recalling the goals we stated in section 1, we argue that we have achieved them. Joining the system is easy because all a node needs is learning about some other active nodes in the system. The bandwidth overhead is reasonably low, in particular because we do not employ cover traffic. Acceptable end-to-end performance is achieved by quickly switching to another tunnel when one offers very poor performance or has stopped working completely.

Based on the assumption that it is extremely unlikely that an adversary is able to monitor the whole system, we have argued that it is only possible for him to occasionally break the anonymity of a user if he manages to observe at least the traffic at the initiator and final node of an anonymous tunnel. As soon as the user switches to another tunnel, her identity is protected again. We have also shown that MorphMix is reasonably resistant to collusion attacks as long as the adversary does not control significantly more than about 20–30% of all participating nodes. Here again, the adversary may fully compromise a few anonymous tunnels, but in most cases, he will fail. Consequently, MorphMix is well suited to protect its users from long-term profiling without guaranteeing the anonymity of every single transaction.

Each node has only to handle its local environment consisting of the peers it is connected to, which is virtually independent of the number of active nodes. Scalability is mainly an issue when determining whether an anonymous tunnel is good or bad. As the time to process a newly arriving selection increases linearly with the system size, MorphMix eventually reaches its limits when the number of nodes approaches 1'000'000.

Compared with static mix networks, MorphMix scales better and is much less vulnerable to legal attacks due to its decentralized nature. The most significant difference is that because of its size and dynamism, MorphMix does not need to employ cover traffic to reasonably protect from traffic analysis attacks, which results in much less overhead.

Our collusion detection mechanism is based on each user's own experience she has collected during the setup of her anonymous tunnels. This is not a problem if the number of participating nodes in the system is relatively small. According to figure 4, it takes about 100 anonymous tunnels of length 5 until reasonable judgments about whether a tunnel is good or malicious can be made, which is an acceptable burden. But with 100'000 nodes, this increases to about 750 and with 1'000'000 nodes to about 8'000 tunnels [16], which is no longer insignificant. Upon joining the system for the first time, the user could either not use anonymous tunnels until she has acquired enough knowledge or always accept tunnels in the beginning and risk frequent observation by a possible adversary. One could also imagine to use the experience of many or all users together that share their extended selections to learn about the system much more quickly. But carelessly giving away the information about extended selection collected during the setup of the own anonymous tunnels could allow others to learn more about these tunnels. In addition, malicious nodes could distribute fake extended selection to confuse well-behaving users. To solve this, one could define a set of trusted witnesses [10] to improve the trust in extended selections received from other nodes.

MorphMix is still very much work in progress and has some limitations in its current state. As anonymous tunnels can fail at any time, the system is best suited for applications making use of several short-lived end-to-end connections such as web browsing. Maintaining longstanding remote login sessions is a problem without being able to reroute anonymous tunnels when a node fails or without making sure that an anonymous tunnel contains only nodes that remain active with high probability. We also do not

yet take prefixes of IP addresses into account because up to now, two IP addresses were either the same or completely different, independent of the number of bits their prefixes match. Taking IP prefixes into account should prevent an attacker from simply operating 1'000 nodes in only a few different subnets or from regularly changing the IP address of a node within its subnet to give it a new identity from time to time. We also have to study peer discovery in more detail to avoid a node mainly learns about malicious nodes forming a collusion, how denial of service attacks can affect the system, and if there are adversarial games that significantly increase the probability to break anonymous tunnels.

Our next steps are to solve these problems to increase the robustness and resistance to attacks, finalize the design of the protocol, and completely implement the system.

## 8. REFERENCES


[1] Adam Back, Ian Goldberg, and Adam Shostack. Freedom 2.1 Security Issues and Analysis. White Paper, `http://www.freedom.net/info/whitepapers/Freedom_Security2-1.pdf`, May 3 2001.

[2] Oliver Berthold, Hannes Federrath, and Marit Köhntopp. Project "Anonymity and Unobservability in the Internet". In *Proceedings of the Workshop on Freedom and Privacy by Design / Conference on Freedom and Privacy 2000 CFP*, pages 57–65, Toronto, Canada, April 4–7 2000.

[3] Oliver Berthold, Hannes Federrath, and Stefan Köpsell. Web MIXes: A System for Anonymous and Unobservable Internet Access. In *Designing Privacy Enhancing Technologies*, volume 2009 of *Lecture Notes in Computer Science*, pages 115–129. Springer Verlag, 2000.

[4] Philippe Boucher, Adam Shostack, and Ian Goldberg. Freedom Systems 2.0 Architecture. White Paper, `http://www.freedom.net/info/whitepapers/Freedom_System_2_Architecture.pdf`, December 18 2000.

[5] David L. Chaum. Untraceable Electronic Mail, Return Adresses, and Digital Pseudonyms. *Communications of the ACM*, 24(2):84–88, February 1981.

[6] Lance Cottrell. Mixmaster Software. `http://www.obscura.com/~loki/remailer/remailer-essay.html`.

[7] Wei Dai. PipeNet. `http://www.eskimo.com/~weidai/pipenet.txt`.

[8] George Danezis, Roger Dingledine, David Hopwood, and Nick Mathewson. Mixminion: Design of a Type III Anonymous Remailer Protocol. Manuscript, `http://mixminion.net`, 2002.

[9] W. Diffie and M. E. Hellman. New directions in cryptography. *IEEE Transactions on Information Theory*, 22(6):644–654, November 1976.

[10] Roger Dingledine, Michael Freedman, David Hopwood, and David Molnar. A Reputation System to Increase MIX-net Reliability. In *Proceedings of 4th International Information Hiding Workshop*, pages 126–141, Pittsburg, PA, USA, April 2001.

[11] Michael J. Freedman and Robert Morris. Tarzan: A Peer-to-Peer Anonymizing Network Layer. In *Proceedings of the 9th ACM Conference on Computer and Communications Security (CCS 2002)*, Washington, D.C., USA, November 2002.

[12] R. Housely and W. Polk. Internet X.509 Public Key Infrastructure. RFC 2528, 1999.

[13] Andreas Pfitzmann and Marit Köhntopp. Anonymity, Unobservability, and Pseudonymity - A Proposal for Terminology; Draft v0.12. `http://www.koehntopp.de/marit/pub/anon/Anon_Terminology.pdf`, June 17 2001.

[14] Michael Reed, Paul Syverson, and David Goldschlag. Anonymous Connections and Onion Routing. *IEEE Journal on Selected Areas in Communications*, 16(4):482–494, May 1998.

[15] Michael K. Reiter and Aviel D. Rubin. Crowds: Anonymity for Web Transactions. *ACM Transactions on Information and System Security*, 1(1):66–92, November 1998.

[16] Marc Rennhard. MorphMix: Peer-to-Peer based Anonymous Internet Usage with Collusion Detection (available at `http://www.tik.ee.ethz.ch/~rennhard/publications/morphmix.pdf`). TIK Technical Report Nr. 147, TIK, ETH Zurich, Zurich, CH, August 2002.

[17] Marc Rennhard, Sandro Rafaeli, Laurent Mathy, Bernhard Plattner, and David Hutchison. An Architecture for an Anonymity Network. In *Proceedings of the IEEE 10th Intl. Workshop on Enabling Technologies: Infrastructure for Collaborative Enterprises (WET ICE 2001)*, pages 165–170, Boston, USA, June 20–22 2001.

[18] Marc Rennhard, Sandro Rafaeli, Laurent Mathy, Bernhard Plattner, and David Hutchison. Analysis of an Anonymity Network for Web Browsing. In *Proceedings of the IEEE 11th Intl. Workshop on Enabling Technologies: Infrastructure for Collaborative Enterprises (WET ICE 2002)*, pages 49–54, Pittsburgh, USA, June 10–12 2002.

[19] Andrei Serjantov, Roger Dingledine, and Paul Syverson. From a Trickle to a Flood: Active Attacks on Several Mix Types. In *Proceedings of 5th International Information Hiding Workshop*, Noordwijkerhout, Netherlands, October 2002.

[20] Ion Stoica, Robert Morris, David Karger, M. Frans Kaashoek, and Hari Balakrishnan. Chord: A Scalable Peer-to-peer Lookup Service for Internet Applications. In *Proceedings of the ACM SIGCOMM '01 Conference*, San Diego, CA, USA, August 2001.

[21] Paul Syverson, Gene Tsudik, Michael Reed, and Carl Landwehr. Towards an Analysis of Onion Routing Security. In *Proceedings of the Workshop on Design Issues in Anonymity and Unobservability*, pages 83–100, Berkeley, CA, USA, July 25–26 2000.

[22] Marc Waldmann, Aviel D. Rubin, and Lorrie Faith Cranor. Publius: A Robust, Tamper-Evident, Censorship-Resistant Web Publishing System. In *Proceedings of the 9th USENIX Security Symposium*, August 2000.

[23] Matt Wright, Micah Adler, Brian Neil Levine, and Clay Shields. An Analysis of the Degradation of Anonymous Protocols. In *Proceedings of ISOC Network and Distributed System Security Symposium (NDSS 2002)*, San Diego, USA, February 2002.