

# 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

Technical Report  
TIK-Nr. 147

August, 2002

## Abstract

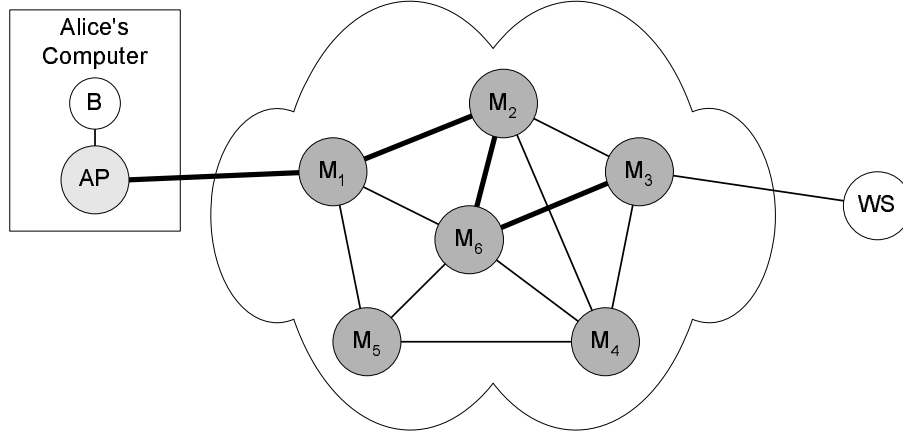
Several MIX-based systems offering anonymity have been operational. They are usually based on a relatively small set of static, well known and highly reliable nodes with good performance and connectivity that offer the MIX functionality. Due to the small number of nodes in such a system, they must be made very resistant against traffic analysis attacks by exchanging lots of cover traffic, resulting in an unacceptable bandwidth overhead. End-to-end traffic analysis attacks are even more difficult to counter as there are only a few entry- and exit-points in the system. Additionally, static MIX-networks suffer from scalability problems and in several countries, institutions operating a MIX could be targeted by legal attacks. In this technical report, we introduce MorphMix, a system for peer-to-peer based anonymous Internet usage. Each MorphMix node is a MIX in the system and anyone with access to a computer connected to the Internet with a public IP address can join the system. We believe that MorphMix overcomes or reduces several drawbacks of static MIX networks. On the other hand, MorphMix introduces new challenges. Particularly, an adversary can easily operate several malicious nodes in the system and try to break the anonymity of a legitimate user by getting full control over an anonymous path she is using. To counter this attack, we have developed a collusion detection mechanism, which allows to identify compromised paths with a very high probability before they are being used.

## 1 Introduction

In 1981, David Chaum proposed the concept of a MIX network [8] 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 service: the mixmaster system [9], Onion Routing [20, 26], Freedom [5, 1], Web Mixes [3, 4], and the Anonymity Network [22, 23].

All these systems follow the same basic principle which is depicted in Figure 1 using a web browsing example.

The core consists of one or more well-known systems, often referred to as MIXes ( $M_1$ – $M_6$ ). If Alice wants to communicate with a web server (WS) anonymously, she uses an access program (AP), which establishes a path via a subset of the MIXes ( $M_1$ ,  $M_2$ ,  $M_6$ , and  $M_3$ ). To exchange data with the web server, her web browser first sends a request to the AP. The AP forwards the data to the first MIX Alice has chosen ( $M_1$ ), from where they are sent to the second and so on, until the last MIX forwards the data



**Figure 1:** Basic MIX network overview.

to the server. The data use the same MIXes in opposite order to find their way back to the user. The goal of a MIX is to hide the correlation between incoming and outgoing messages such that an attacker that performs traffic analysis to break the anonymity cannot follow a message through the network. Such a system of MIXes is usually called a *MIX network* and makes use of the following basic measures to thwart traffic analysis:

1. All messages used in the system have exactly the same length. This defeats traffic analysis based on correlating packets by their length.
2. The encoding of a message has to be changed between entering and exiting a MIX. This is usually achieved by encrypting or decrypting a message as it passes a MIX. This prevents an attacker from successfully performing traffic analysis by simply looking at the content of packets.
3. A MIX delays and reorders incoming packets from different users before forwarding them such that the outgoing sequence of packets is not related to the incoming sequence. This makes traffic analysis based on timing very difficult because for an eavesdropper, each of the outgoing packets can correspond to each of the incoming packets with the same probability.
4. A MIX must guarantee that no message is processed more than once. This hinders an attacker from successfully carrying out a replay attack by re-sending a message to a MIX and comparing the outputs. If he observes the same output twice, then he can correlate an incoming and outgoing message.

The goal of most MIX-based systems is to offer *sender anonymity* and *relationship anonymity* [17]. This means that the receiver (or server) should not find out who the sender (or client) is and an eavesdropper should not be able to detect that there is a communication relationship between the two parties. Although there are applications for receiver anonymity, e.g. anonymous Web publishing [27], most Internet activities where anonymity is desired require only sender and relationship anonymity. Sender and relationship anonymity are also the basis for pseudonymous applications, where a user contacts a well-known service using a pseudonym instead of her real identity. An example for such an application is pseudonymous e-commerce [19].

## 2 Analysis of the Current Model

In this section, we analyze the properties of the current model as described above.

## 2.1 Benefits

The current model offers several benefits, the most dominating ones are briefly described in the following paragraphs.

- **Accessibility.** Using well-known MIXes makes it straightforward for a user to access them. Their identities (host names or IP addresses) can be made public through web sites or news messages in the Usenet and the addition and removal of MIXes can also be made public in this way. In general, it is not expected that MIXes appear and disappear frequently. Rather, the model assumes that each MIX remains operational for a long time (e.g. months or years) before it terminates its service.
- **Authentication.** Using digital certificates [15] makes it possible to control which MIXes are allowed to offer their services. This prevents unauthorized (and potentially malicious) MIXes from joining the system and collecting information and in addition gives the user the possibility to identify and authenticate the MIXes she is using. A centralized system could play the role of the certification authority responsible for issuing certificates for MIXes. This does of course not guarantee that malicious MIXes are present in the system, but gives at least the possibility to evaluate the trustworthiness of a MIX before certifying it.
- **Reliability.** By controlling who is allowed to operate a MIX, one can make sure that only highly reliable MIXes are present in the system. In addition, there could be minimal requirements for the computing power of the MIXes and for the network connection they offer in order to make sure that no particular MIX becomes a bottleneck in the system.

To summarize, smooth operation of a system based on the current model is relatively simple because of the limited set of highly reliable MIXes that offer good computing power and have good network connectivity.

## 2.2 Limitations

On the other hand, several limitations exist with the current model:

- **Scalability.** As the number of MIXes is relatively small (e.g. several hundreds) compared to its potential number of users (e.g. hundreds of thousands), the system eventually reaches its limits with respect to the traffic it can handle. Of course one can simply add more MIXes to the system to extend its capacity, but in general this drastic imbalance between MIXes and system users poses a significant problem.
- **Attacks on a MIX.** Like a legitimate user, an attacker can also easily find out the list of static MIXes the system consists of. If the attacker can passively observe all traffic that enters and leaves a MIX, he can perform traffic analysis to correlate incoming and outgoing messages at that MIX. A MIX can prevent this attack from being successful by reordering and delaying messages and by using dummy traffic that are exchanged between the MIXes. However, delaying messages at each MIX also means a decrease in the end-to-end performance the user experiences, which especially is a problem if near real-time application such as web browsing make use of a MIX network. In addition, it is not clear today if this attack can be effectively prevented without employing a constant data rate where messages are exchanged all the time between two MIXes. This results in vast amounts of dummy traffic overhead which seems unacceptable in today's Internet.
- **End-to-End traffic analysis.** Another attack tries to correlate events at the endpoints of the system: if a user makes an HTTP request, it is reasonable to assume that this request leaves the last MIX towards a web server shortly later. Similarly, the response sent from the web server to the

last MIX will appear on the link between first MIX and user within some seconds. The user could only be protected from this attack if a constant traffic flow is always established between the user and the first MIX. Considering the many users one MIX would have to handle at the same time, a lot of bandwidth of this MIX would be absorbed just by the dummy traffic sent to and received from these users.

- **Legal attacks.** There are governments and other institutions that do not like the idea of anonymity in the Internet. While technical attacks on the system may not easily work out, legal attacks could be quite simple (also known as rubber-hose cryptanalysis). Assume a university operates a MIX: the government just contacts the key persons at that university and threatens them to stop all current and future funding of their research if the service offered by the MIX is not discontinued within one week. In general, any system with centralized components or services is very vulnerable to legal attacks.

To summarize, the current model does not scale well because of the large number of potential users compared to the relatively few MIXes. In addition, there exist several attacks on the system, mainly because the identities of the MIXes is well known to legitimate users and attackers. Especially the legal attack seems to be a big threat and is difficult to defend against.

### 3 An Alternative Model based on Peer-to-Peer Technology

In this section, we present the basic concept of the MorphMix system. It is based on an alternative model to enable anonymous communication in the Internet. We do no longer distinguish between clients (end-users) and servers (MIXes). Rather, each end-user is also a MIX at the same time - all participants are equal peers.

#### 3.1 Motivation and Goals

The motivation for this paradigm shift is that we believe a peer-to-peer environment is better suited for offering anonymity in the Internet. Especially, we believe that the drawbacks of the current model can be reduced or eliminated. Since every user is a MIX, scalability issues should not arise, because the capacity of the whole MIX network is no longer fixed, but is increased with each joining peer. Legal attacks on the system are hardly successful as there is no centralized institution to attack. 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. Additionally, nodes will appear and disappear and the whole system will be dynamic and change continuously, which makes it virtually impossible for anyone to get knowledge of the full network at any time. As a result, is no longer practical for an attacker to perform passive traffic analysis attacks at each MIX. Similarly, end-to-end traffic analysis attacks are more difficult to carry out, first of all because there are so many possible exits for each path through the system and second, because there is no longer easily identifiable entry-points into the system (the link between the end-user and the first MIX) as in the traditional model. In the new model, if a peer B gets traffic from another peer A, then B cannot tell if the traffic originated at A or if A is simply relaying the data from yet another peer.

However, the alternative model also introduces many new challenges. One is that we do no longer have a stable set of highly reliable MIXes, but rather a dynamic system of unreliable peers that may join and leave the system at any time. In addition, some peers have good network connectivity while others employ only slow dial-up connections. We also do not rely on a global public key infrastructure (PKI) since this will not be realized in the near future and since we want each user be easily able to run a peer. This implies that it is not possible to unambiguously authenticate the other peers one is communicating with, and also that it is easy for a malicious node (and multiple colluding nodes) to join the system.

We move from a stable system consisting of a small set of highly reliable MIXes that offer good performance and connectivity to a highly dynamic system with a large set of unreliable peers that may join and leave at any time. In addition, we move from a system where each MIX is well known, can be authenticated and is probably not malicious to a system where unambiguously authenticating a peer is not possible and where malicious peers can easily join.

The main goals of the MorphMix system are the following:

- **Anybody can easily join.** MorphMix should allow any computer with a public IP address that is connected to the Internet to participate, regardless of the bandwidth of its network connection and its availability. MorphMix should not be a system that only allows highly available nodes with good network connectivity to join.
- **Keeping the bandwidth overhead low.** MorphMix should try to keep the bandwidth overhead as low as possible while offering a high level of anonymity. In particular, we do not want to use cover traffic because keeping up cover traffic flows between nodes in a way that really protects from traffic analysis attacks without significantly degrading the end-to-end performance is very difficult in an environment with unreliable nodes.
- **Protection from collusion attacks.** Due to its openness, an adversary can operate several malicious nodes and try to break the anonymity of legitimate users. It therefore may happen to a user most nodes along the path she is using to communicate anonymously are controlled by that adversary. MorphMix should make it possible to detect such paths with high probability before any data are sent through it.
- **Protection from traffic analysis attacks.** MorphMix should make use of the dynamic nature of the system and enforce that nodes frequently change the peers they are communicating with.
- **Offering acceptable end-to-end performance despite the dynamic environment.** Allowing any node to join regardless of its reliability means nodes will suddenly stop operating without prior warning. A node terminating its service causes all paths using that node to fail. MorphMix should cope with such situations as well as possible.
- **Scalability.** The number of nodes MorphMix consists of is open-ended. MorphMix should be able to efficiently handle potentially millions of nodes.

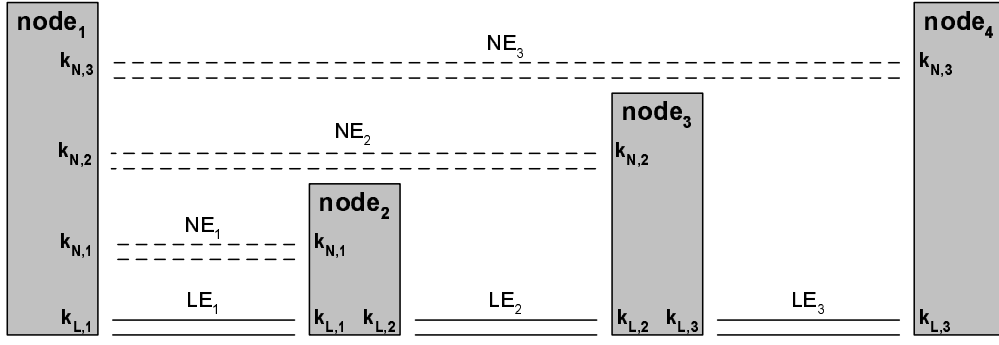
## 3.2 Basic Design

In this section we describe the design of MorphMix. It consists of an open ended set of nodes. A node  $i$  is identified by its IP address  $ip_i$ . In addition, each node has a key-pair consisting of a private key  $PrK_i$  and a public key  $PuK_i$ . This key-pair is generated locally by the node when it is started up for the first time. To access other nodes,  $i$  must know their IP addresses and public keys.

To access the Internet anonymously, a user sets up an *anonymous tunnel*, which starts at her own node, via some other nodes. We explain the concept of an anonymous tunnel in section 3.3. We name the node that is setting up the anonymous tunnel the *initiating node* or simply the *initiator*. The last node of the anonymous 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 collude with other nodes to break a particular user's anonymity and *malicious nodes*.

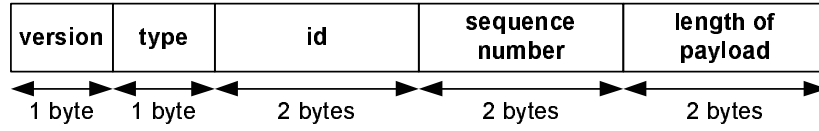
## 3.3 Message Forwarding

Figure 2 depicts a fully set up anonymous tunnel from node<sub>1</sub> via node<sub>2</sub>, node<sub>3</sub>, and node<sub>4</sub>. Node<sub>1</sub> is the initiator, node<sub>2</sub> and node<sub>3</sub> are intermediate nodes, and node<sub>4</sub> is the final node.



**Figure 2:** *Layers of encryption.*

We make use of layered encryption similar to the approach proposed by Chaum [8]. We denote by  $\{m\}_k$  the encryption of message  $m$  with key  $K$ . When node<sub>1</sub> sends a message  $m$  through the anonymous tunnel, it encrypts it repeatedly with the keys corresponding to the *nested encryptions* (NEs), which results in  $\{\{\{m\}_{k_{N,3}}\}_{k_{N,2}}\}_{k_{N,1}}$ . To identify the messages belonging to a particular anonymous tunnel, each message needs a unique tag on each link between two nodes. We therefore prepend a header to the encrypted message. The header is depicted in figure 3.



**Figure 3:** *Message header.*

The *version* and *type* fields are defined as unsigned 8-bit integers. The various types are needed to interpret the messages correctly, i.e. to distinguish setup messages from normal data messages. The *id* is an unsigned 16-bit integer and is needed to multiplex anonymous tunnels on a link between two proxies. The *id* has only local significance on the link between two proxies, and the *id* changes when a message traverses a node. If a message with *id*  $a$  arrives on link  $A$ , then the AP has to know on which link and with what *id* to forward it. So all a node needs to know to handle a message correctly is a mapping  $link_A: id_a - link_B: id_b$ . The *ids* on each link are chosen during the setup of the anonymous tunnel and remain the same until it is torn down. The unsigned 16-bit integer *sequence number* is needed to prevent replay attacks on a node and is incremented for each new message. As each anonymous tunnel will be active only for a relatively short time (see section 3.7), 16-bits are enough without having to reuse sequence numbers in the same anonymous tunnel. Finally, the unsigned 16-bit integer field *length of payload* identifies the number of bytes in the following payload field. This allows messages being padded to a constant size.

Before node<sub>1</sub> sends the repeatedly encrypted message to node<sub>2</sub>, it prepends the header with the appropriate values, encrypts the header according to the *link encryption* (LE) between node<sub>1</sub> and node<sub>2</sub> using the key  $k_{L,1}$  and sends it to node<sub>2</sub>. Node<sub>2</sub> receives the message, decrypts the link encryption using  $k_{L,1}$ , removes one layer of encryption using  $k_{N,1}$  which results in  $\{\{m\}_{k_{N,3}}\}_{k_{N,2}}$ . Using the *id* in the message header, node<sub>2</sub> determines the next link and the *id* to use on that link, prepends a message header with the correct values, encrypts the header according to the link encryption between node<sub>2</sub> and node<sub>3</sub> using the key  $k_{L,2}$  and sends it to node<sub>3</sub>. This continues until the final node is reached who relays the data to the server node<sub>1</sub> wants to communicate with. Messages are sent back to node<sub>1</sub> in the same way but in opposite order. This time, each node adds a layer of encryption instead of removing one.

To multiplex multiple *anonymous connections* within an anonymous tunnel, a slightly modified message header is used. It corresponds to the one in figure 3 but without the sequence numbers and results in a total length of 6 bytes. This header is prepended to messages sent from the first to the last node of

an anonymous tunnel before the nested encryptions are applied. It is important to realize the concept of multiple anonymous connections that are multiplexed within one anonymous tunnel. The anonymous connections have only a meaning for the first and last nodes of an anonymous tunnel. The messages themselves are routed through the system using only the anonymous tunnel information.

### 3.4 MIX Functionality

In section 1, we listed the measures employed by MIX networks to thwart traffic analysis. We will make use of most of them in MorphMix. All messages exchanged between nodes have the same length. The nested encryptions guarantee that messages change their coding when traversing a node. The sequence number in the message header defeats replay attacks. However, the delaying and reordering of messages is possible only in a very limited way. Considering that MorphMix should be useful for near real-time applications such as web browsing, the time a message is stored in any node on the way from the server to the initiator should be minimized in order to get acceptable end-to-end performance. To biggest difference compared to static MIX networks is that MorphMix will not use dummy traffic between nodes. While the design of the system would allow for dummy messages, we believe that they would be too big a burden especially for those nodes that have very limited bandwidth to share. However, as already mentioned in section 3.1, the resistance of MorphMix is not based on protecting each node as well as possible, but on the assumption that observing the vast number of nodes and the links between them is not feasible. As a consequence, dummy traffic would add only little to protect the system even further.

### 3.5 TCP or UDP?

An important question to answer is whether the data sent between nodes uses TCP or UDP. Both protocols have their advantages and disadvantages when used in a MIX network.

#### 3.5.1 TCP

When the TCP protocol is used between a pair of MIXes, the user's application usually accesses the anonymizing network in the same way a web browser accesses a web proxy: a TCP-connection is set up to the access program running on the user's computer, which in turn handles the communication with the MIX network. When the data travels through the network, it is sent across TCP-connections on each link between two adjacent MIXes. To function properly, the access program usually needs to understand the protocol of each application it provides access. For instance, if it is accessed by a web browser, it needs to know part of the HTTP protocol [2] to interpret the various methods such as GET or CONNECT correctly. The disadvantage of this is that the access program has to be extended whenever a new application should be supported. On the other hand, this approach makes it quite easy to support different platforms as the access program runs only on the application level accessing the socket interface without requiring special privileges.

Using TCP-connections implies that the properties of TCP – flow control and correct delivery of all data in the right order – are guaranteed hop-by-hop and not end-to-end between the user's application and the server. Consequently, a MIX must not lose any data of an end-to-end connection or the application will usually fail. On the other hand, TCP makes the reliable communication between MIXes that have very different bandwidth connections quite easy because the transport layer takes care that all data is delivered correctly.

When a packet of a TCP-connection is lost, then every end-to-end connection using that particular link stalls. In an environment with relatively few static MIXes and consequently few connections between a pair of MIXes that each carry many end-to-end connections, this could be a potential performance problem. However, in a highly dynamic environment with very many MIXes, this shouldn't be a major problem, since a connection between two MIXes never carries very many end-to-end connections.

### 3.5.2 UDP

When UDP is used between a pair of MIXes, the approach is usually quite different: an application communicates directly with the server and is unaware of the MIX network. On the user's computer, data is extracted after the IP-layer and – after removal of information that could possibly identify the user's computer such as its IP-address – inserted into a UDP datagram. This data is then sent through the MIX network, hop-by-hop. Note that since data has to be extracted from the IP-stack, i.e. from the kernel space, different operating systems have to be specially supported. Additionally, extracting data from the IP-stack is usually not possible without special privileges.

With UDP, there are also no guarantees that the data gets through the MIX network. If the application uses TCP, the endpoints of the connection are the user's computer and the server: if a UDP datagram is lost on the way from the server to the user's computer, the TCP-layer in the server eventually realizes that no acknowledgement has arrived and therefore retransmits the data.

An advantage of using UDP between MIXes is its transparency to the end-to-end transport and application protocols. Therefore, customization to a particular application is not needed. Using UDP also solves the potential performance problem imposed by stalled TCP-connections in a static MIX network with only a few MIXes. However, using UDP is also dangerous since a fast sender can easily overload a slow receiver, which results in large amounts of lost datagrams. UDP makes sense in an environment where all the MIXes have similar computing power and bandwidth. In such an environment, each link between two MIXes can be tuned to its maximum throughput without having too many lost datagrams. However, it is probably very difficult to achieve the same in a dynamic and heterogeneous environment when using UDP.

Taking into account the advantages and disadvantages of the two protocols, we have decided to use the TCP protocol to communicate between two nodes. The reasons for this decision are the following:

- It should be possible to use MorphMix on a variety of operating systems without special privileges. This is easier with TCP than with UDP, as no data has to be extracted from the kernel space.
- Taking into account the heterogeneity of the nodes participating in our network, using TCP makes life much easier. With UDP, two nodes would have to employ some sort of flow control between them to achieve acceptable performance without losing too many packets. It is questionable if one could do much better than using TCP directly.
- As we will make use of fixed-length messages between nodes to improve the protection from attacks, UDP would waste a lot of bandwidth: since the ACK messages of the end-to-end TCP-connections are transported within UDP datagrams, the effective payload of them is only short and the datagrams would have to be padded to the fixed-length of messages. With TCP, this problem does not occur as the TCP messages only transport application data and there are no application-level ACKs.

It should be noted that our collusion detection mechanism presented in section 4 works independently of using TCP or UDP between the nodes.

### 3.6 Who chooses the next hop?

Another important design decision is who chooses the path through the network. There are basically two options: (1) the initiating node picks all nodes by itself or (2) the initiator only picks the first hop, whereas each node along the anonymous tunnel then picks the following hop. We briefly compare the two approaches here.

If the initiator picks all the hops it is using in an anonymous tunnel path, it must know about at least these nodes (and their public keys) beforehand. In addition, it has to know if the selected hops can offer

enough resources to satisfy the throughput it is expecting or if they are actually willing to accept new anonymous tunnels. In general, this requires some sort of lookup-service which is so up-to-date that if a user queries for information about other nodes to use in her anonymous tunnel, setting up the tunnel will succeed with a high probability. There exist scalable peer-to-peer lookup services such as Chord [24], but we believe that the overhead imposed by such a service is still significant and we therefore choose not to make use of a lookup service.

Letting the initiator choose the whole path has more drawbacks: to complicate traffic analysis, it is reasonable that each node is connected to a limited number of other nodes. This guarantees that a TCP-connection between two nodes carries several anonymous tunnels that are not visible from the outside (due to the encrypted message headers, as described in section 3.3. If an initiator selects the whole path, then it also determines to which next hop each node along the anonymous tunnel has to establish a connection. Changes are that a node must connect to another node for each new anonymous tunnel it is involved in, which means that each TCP-connection between a pair of nodes carries the data of only one anonymous tunnel.

We have decided to let each hop pick the next hop in an anonymous tunnel by itself. Besides solving the problems above, this has additional benefits:

- Each node can decide to which other nodes it utilizes connections. For instance, it might be reasonable for a node to mainly establish connections to other nodes that are relatively close.
- Similarly, adjacent nodes can communicate with each other and exchange information about the bandwidth they can offer or if they are willing to accept further anonymous tunnels. This allows a node to balance the load among its neighbors.

There is one problem when we allow each hop to pick the next hop: once we hit a malicious node that wants to collect data about anonymous tunnels, we can expect that this node either simulates all remaining hops by itself or that it uses an accomplice as the next hop. This problem does not occur if the user selects all hops by herself and gets the hops and their public keys from a lookup-service. In that case, the user can use the public keys of the nodes to encrypt the keying material for the nested encryptions to make sure that only they can decrypt it. We'll show in section 3.9 how to solve this.

### 3.7 Dynamism of MorphMix

We have mentioned in section 3.1 that the resistance of MorphMix to attacks mainly stems from the vast number of possible paths an anonymous tunnel can take and its dynamism in the sense that the peers the system consists of are changing continuously. However, to take full advantage of the big number of nodes, it is not enough to discover some peers once a node has become active and communicate with them for hours, as this would greatly limit the possible previous and next hops of anonymous tunnels through that node during the time it is active. Rather, each node should constantly try to learn about other peers that can be used as a possible next hop node in anonymous tunnels.

We assume that 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. Using this link, the two 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 exchanging 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. However, to guarantee that no link between two nodes is used for too long, no new anonymous tunnels will be set up within a link after a while. When all anonymous tunnels within that link are terminated, the link between the two nodes is torn down and they are disconnected again. A node always is connected to several other nodes, which implies that looking

for other nodes, establishes a link encryption with them, using each link for a while, and tearing them down again is an ongoing process.

Similarly, when acting as the initiator of anonymous tunnels, a node does not establish one anonymous tunnel and uses it for a while, but keeps setting up anonymous tunnels in the background. The goal is to have some anonymous tunnels established at any time that could be used to communicate anonymously. Each anonymous tunnel is only used for a relatively short time and several can be used in parallel, if the application makes use of more than one end-to-end connection at a time (think about communicating with a web server and receiving the embedded objects within a page through various TCP-connections). Having more than one anonymous tunnel available at any time also 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.

Using this combination of several, but short-lived anonymous tunnels and the continuous change of peers a node communicates with, we believe we make it very difficult for an attacker to perform traffic analysis. In addition, when discussing the collusion detection mechanism of MorphMix in section 4, we will see that it is heavily based on the dynamism of the system.

### 3.8 Discovering Other Nodes

We have mentioned above that we do not employ a lookup-service that keeps track of all currently active nodes. Nevertheless, we need a way to learn about some other nodes. There are different ways to do so: to join, one must learn about 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, hoping that one of them is also active now. Another way to learn about other nodes is to query some nodes that are known to be always up. A third way is that there are some information servers that know about “many” currently active nodes. With many nodes, we mean that each of these servers knows several nodes but it does not care about what percentage of all nodes it actually knows. Each participating node contacts one of these servers from time to time and tells it about the nodes it currently knows and in return gets some other active nodes from the server. 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, and this is exactly what we need to keep the system dynamic.

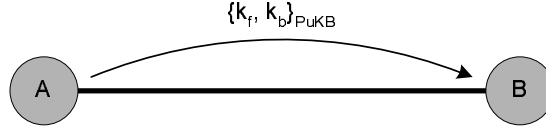
### 3.9 Anonymous Tunnel Setup

In this section, we describe the setup of an anonymous tunnel and analyze the possible attacks to break a user’s anonymity.

#### 3.9.1 Setting up the Link Encryption

When a node A wants to set up the link encryption with another node B, it first establishes a TCP-connection with B. If B does not want to accept further connections at this time, it simply stops listening on the corresponding port temporarily. If B accepts the connection, A picks two random bit-strings that serve as the forward- and backward-keys on that link. The forward-key is used to encrypt subsequent data sent from A to B and the backward-key to encrypt data sent from B to A. The keys will be used in a symmetric cipher and - depending on the cipher used - should be at least 128 bits long. Figure 4 illustrates the setup of a link encryption.

If  $k_f$  denotes the forward-key,  $k_b$  the backward key, and  $\text{PuK}_B$  B’s public key, then the key-exchange message sent from A to B is  $\{k_a, k_b\}_{\text{PuK}_B}$ .



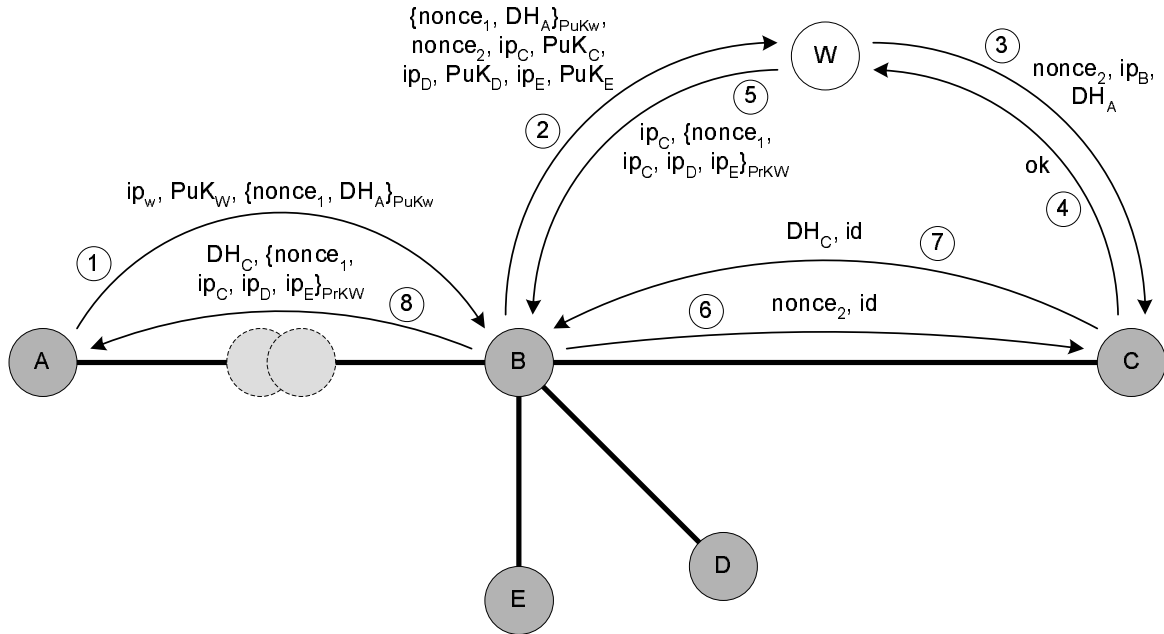
**Figure 4:** *Setting up the Link Encryption*

### 3.9.2 Setting up the Nested Encryption

The setup of a nested encryption takes place between the initiating node and a node along the anonymous tunnel. Since the initiator does not know the nodes along its tunnel beforehand (except the node following immediately the initiator), we cannot exchange keys simply by encrypting a symmetric key with the recipients public-key as we did for the link encryptions. We therefore use the Diffie-Hellman (DH) [11] key-exchange.

If the initiator simply sent its half of the DH key-exchange to node B responsible for selecting the next hop C, node B could easily play the role of node C (and other nodes following C) itself without the user noticing this. Similarly, node B could select a next hop node C as requested, but carry out a man-in-the-middle attack as each following message traverses C. The problem we have is that if B is malicious and sees the initiator's half of the DH key-exchange in the clear, then no matter how many nodes the user specifies to use after B do not increase the anonymity.

To solve this problem, we introduce the notion of a *witness*. For each hop, a witness is selected by the initiator. The witness should be selected randomly among the nodes the initiator 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 5 illustrates the procedure to set up one hop of the anonymous tunnel.



**Figure 5:** *Setting up the Nested Encryption*

Node A is the initiator setting up an anonymous tunnel. We assume the tunnel has already been set up to node B (via one 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 a the nested encryption to 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 ( $DH_A$ ) and a nonce  $nonce_1$  which is needed to prevent replay attacks.  $nonce_1$  and  $DH_A$  are encrypted using W's public key  $PuK_W$ , which results in  $\{nonce_1, DH_A\}_{PuK_W}$ . A then sends a message to B through the formerly established part of the anonymous tunnel consisting of W's IP address  $ip_W$ , W's public key and the encrypted nonce and DH parameters. The message tells B to append a node to the anonymous tunnel using the witness W.

2. B receives the message and establishes a link encryption to W, using  $ip_W$  and  $PuK_W$ . It generates a nonce  $nonce_2$ , which is used to recognize later messages. B generates a message containing the encrypted nonce and DH parameters from A,  $nonce_2$ , the IP addresses of potential next hop nodes ( $ip_C$ ,  $ip_D$ , and  $ip_E$ ) and their public keys ( $PuK_C$ ,  $PuK_D$ , and  $PuK_E$ ) and sends it to W.
3. W receives the message and picks randomly one node from the nodes offered by B as the next hop. In figure 5, W picks node C and establishes a link encryption with C using  $PuK_C$ . W also decrypts  $nonce_1$  and  $DH_A$  using its private key  $PrK_W$ , generates a message consisting of  $nonce_2$ ,  $ip_B$ , and  $DH_A$ , 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 selection of IP addresses that was offered by B to W and is signed by W using its private key  $PrK_W$ . The first IP address in the receipt is the one W has picked as the next hop. The receipt also contains  $nonce_1$  to guarantee its freshness. W generates a message consisting of the receipt and C's IP address and sends it to B.
6. B receives the message from W and now knows that W has selected C as the next hop. It generates a message containing  $nonce_2$  and the id to be used to identify data belonging to this anonymous tunnel on the link between B and C.
7. C gets the message and sends its part of the DH key-exchange  $DH_C$  back to B via a message identified with id.
8. B generates a message consisting of  $DH_C$  and the receipt that was issued by W and sends it to A.

If C does not accept further anonymous tunnels after receiving message 3, it sends a nok-message during step 4. W then simply repeats step 3 using another node of those specified in message 2. Note however that in this case, node C will not be included in the receipt generated by W. If none of the nodes specified by B is willing to accept the anonymous tunnel, W sends a nok-message to B in step 5. B itself then sends a nok-message to A and the anonymous tunnel is torn down.

Note that the same procedure is used to add the hop directly following the initiator A. Of course, A could simply pick the next hop 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 and this is something we want to avoid.

Before analyzing the attacks, we recall 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 first encrypting  $DH_A$  for W, sending it to W via B (messages 1 and 2), decrypting it at W, and only sending it in the clear from W to C (message 3). B never sees  $DH_A$  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 therefore 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 B's selection must be in that collusion.

### 3.9.3 Analysis of the Nested Encryption Setup

As discussed in section 3, it is quite possible for malicious nodes join the system to either disturb the service offered or to learn more about anonymous end-to-end connections. We discuss here the second group: a single malicious node or a set of collaborating nodes join the system to break the anonymity it offers to its users. We are using figure 5 as the reference for the analysis: node A is the initiator of the anonymous tunnel, B is the currently last hop, W is the witness to choose the next hop, and C is the next hop chosen by W.

**Case 1: B is malicious** Let's assume W is honest but B is malicious. In addition, assume that no further nodes are collaborating with B. B's goal is to break the nested encryption that will be set up between A and C to read the data exchanged between them. To do so, B can either simulate the next hop by itself or carry out a man-in-the-middle attack on the DH key-exchange between A and C. In the first case, the attack works as follows:

1. In message 2, B replaces the public keys corresponding to the IP addresses with self-generated versions it knows the private keys of. The IP addresses can be those of existing, active nodes or those of non-participating or non-existing nodes.
2. B intercepts the setup message for the link encryption sent from W to C to learn  $DH_A$  in message 3.
3. B generates the ok-message sent from C to W in message 4 to get the receipt from C in message 5.
4. B generates C's half of the DH key-exchange and inserts it in message 8.

B is now simulating C itself and A has no way to find out about this. Similarly, B can continue to carry out the same attack for the hops to come, which means that B simulates all remaining hops of the anonymous tunnel. This would be the same as if A had not chosen to use additional hops after B. If B wants to use a real node C as the next hop while carrying out a man-in-the-middle attack, B does the following:

1. In message 2, B replaces the public keys corresponding to the IP addresses with self-generated versions it knows the private keys of. This time, the IP addresses must be those of active nodes.
2. B intercepts the setup message for the link encryption sent from W to C and performs a man-in-the-middle attack on that link. B re-encrypts the setup message for the link encryption (see figure 4) using C's real public key. As B knows the encryption keys, it can intercept message 3 and gets  $DH_A$ .
3. B replaces A's half of the DH key-exchange ( $DH_A$ ) with a self-generated version  $DH_A'$  and sends it to C in message 3. The protocol then continues normally until B receives message 7.
4. Before sending message 8 to A, B replaces C's half of the DH key-exchange ( $DH_C$ ) with an own version,  $DH_C'$ .

B has now broken the nested encryption between A and C, since it has split it into two parts: between A to B, the data is encrypted using keys generated from  $DH_A$  and  $DH_C'$ , whereas between B to C it uses keys generated from  $DH_A'$  and  $DH_C$ . In contrast to the case where B simulates the next hop C by itself, it is now much more difficult for B to control the hops following C, as setting up the next hop will be handled by C. Assume that the next witness is V and the next hop selected by V is F. One possible attack for B would be to replace  $PuK_V$  (in message 1) with a fake public key it knows the private key of. B would then intercept the link encryption setup message from C to V, read the encryption keys and regenerate the message for V, this time using V's real public key. B could then also intercept message 2

and replace the public keys belonging to the IP addresses with own public keys. The next step would be to intercept the link encryption setup message from V to F, read  $DH_A$  from the following message 3, and the attack continues similarly as described above. However, we point out that this attack is much more complicated to carry out than the one where B simulates all remaining hops by itself because this second attack requires active control of many more links than the first. We therefore concentrate on the first attack as it is the simpler one and therefore more likely to be carried out.

How realistic is the first attack? The main difficulty is that B needs active control over the link between W and C with the capability to intercept and inject data packets. Since B cannot predict which witness A is going to choose, 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. For instance, B could only list IP addresses that are in the same subnet as B. It is then relatively easy for B to access the data packets exchanged between W and C. To tackle this attack, we require that the all IP addresses offered by B and B's own IP address must not have similar IP prefixes. This does not prevent the attack, but makes it much more difficult as B needs active control over several links from W to the nodes it offered to carry it out successfully.

Another question is the number of nodes B has to specify in message 2. If B simply specifies one node, then it knows which node W is going to select as there is only one to choose from. Therefore, we have decided to let A choose the number of nodes that B must offer to W. If the number is high, then it is more difficult for B to carry out the attack, but on the other hand a legitimate B may not be able to offer so many nodes as next hops, which causes the setup of the anonymous tunnel to fail. If the number is low, then the probability that the anonymous tunnel can be set up high but it is also easier for B to cheat. We will return to this discussion in section 4.4.

We conclude that a powerful attacker having active control over several links might be able to carry out the attack described above and simulate the remaining hops of an anonymous tunnel.

**Case 2: W is malicious** If W is malicious, then W itself can carry out a man-in-the middle attack by first replacing  $DH_A$  in message 3 with a self generated  $DH_A'$ . Similarly, W subsequently intercepts message 7 on the link between B and C and replaces  $DH_C$  with its own version,  $DH_C'$ . W has then broken the nested encryption between A and C, but is still not able to read the messages exchanged between B and C due to the link encryption. To do so, W would have needed to carry out a man-in-the-middle attack when the link encryption between B and C had been set up. Since a node cannot predict when it will be selected as a witness and especially not what node it will have to witness, there is no way for a node to really prepare for this attack. We therefore conclude that this attack is very unlikely to succeed.

Another option for W is to impersonate C and intercept and reply to all messages sent from B to C. This again requires W to have powerful active control over network links, and again it only works if W already has control over the link encryption between B and C. The advantage to the scenario where W is acting as the man in the middle is that it gets somewhat easier for W to compromise the following hops as it could try to simulate the remaining hops (see case 1 above).

**Case 3: B and W are in the same set of cooperating malicious nodes** In this case, it is trivial for B to simulate the next hop C itself, since W can decrypt  $DH_A$  in message 2 and tell it to B. In addition, W can sign IP addresses at will in message 5. However, since A chooses randomly a different witness for each link to be set up, the probability that all witnesses are cooperating with B is quite small if we assume that only a relatively small portion of all nodes are cooperating with B. As soon as the witness for a link is not cooperating with B, it gets difficult again for B to simulate the next hop (see case 1 above).

**Case 4: B is part of a set of cooperating malicious nodes** If we assume that B is not alone but is part of a larger set of cooperating malicious nodes, then B can simply list 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 have required 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.

Summarizing the attacks discussed in this section, we conclude that the most realistic attack is the one where a set of cooperating malicious nodes tries to learn more about anonymous end-to-end connections (case 4). The first two attacks (cases 1 and 2) require active control over several network links and are therefore much harder to carry out. In addition, if something like a world-wide PKI ever got deployed, then the use of digital certificates would defeat the man-in-the-middle attacks in cases 1 and 2 because impersonating another party would no longer be possible if C signed message 7. There is not much we can do to defend against the third attack (case 3), but we already mentioned above that if it indeed happens that B and W are colluding, then only one leg of the anonymous tunnel is broken. In addition, using certificates and having C sign message 7 would make it impossible for B to impersonate C, but it still would not be needed for B to offer a selection to W in message 2. B could simply use the node it wishes as the next hop C and W would produce the signature on a fake selection, which could contain any IP addresses besides the one of C. Controlling all legs of an anonymous tunnel – even if certificates would be used – is easiest using the attack described in case 4. In the next section, we discuss how we can cope with this attack.

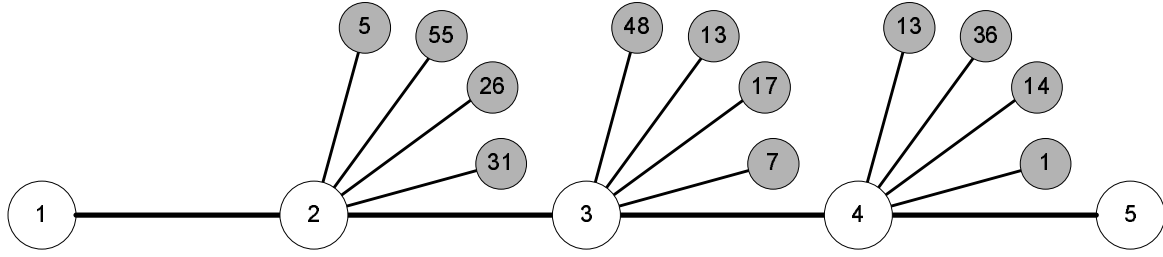
## 4 Preventing Attacks from Cooperating Malicious Nodes

In section 3.9.3, we have seen that the most realistic attack stems from a set of cooperating malicious nodes. In this section, we show how we can counter this attack to minimize its effect.

In general, it is a hard problem to detect nodes that are just collecting data but otherwise offer good service. The initiator or any of the nodes preceding the colluding nodes cannot tell if they are maliciously collecting data or not. However, there is one key difference between an anonymous tunnel that was set up via well-behaving nodes from one that is partly composed of cooperating malicious nodes: In the first case, each node is selected more or less randomly from all nodes participating in the system at the time the tunnel was set up, while in the second case, nodes from the malicious set appear with higher probability. Recognizing nodes that appear more often together in anonymous tunnels can only work when a user has set up and used a variety of different anonymous tunnels. It is therefore reasonable that a user makes use of each anonymous tunnel for only a relatively short time. This guarantees the following: (1) the user will use several different anonymous tunnels and can therefore potentially detect cooperating malicious nodes, since they appear together in one anonymous tunnel with high probability and (2) even if an anonymous tunnel contains several nodes of the cooperating set, the user will only use it for a short time.

In this technical report, we describe the basic collusion detection model. This basic model does not yet take prefixes of IP addresses into account. If all bits of two IP addresses match, then they are the same. If at least one bit differs, they are different. The collusion detection is based on the receipts a user gets from different witnesses during the setup of anonymous tunnels (figure 5 messages 5 and 8). The receipt contains the selection of possible next hops offered to the witness (figure 5 message 2). Since the first node in the receipt is the one selected by the witness, the user knows which node has offered what next hop nodes for each intermediate node in her anonymous tunnel. We name the set of nodes a node offers as possible next hop nodes the *selection* of that node. For now, we will identify the nodes with numbers. Figure 6 illustrates the selection of nodes a user gets when setting up an anonymous tunnel.

Node 1 sets up an anonymous tunnel via 4 other nodes (2, 3, 4, and 5). Each of the intermediate nodes sends a selection back to node 1. For instance, node 2 has offered the nodes 5, 55, 26, 31, and 3 as possible next hop nodes. Since the witness always puts the node it has selected first, node 1 gets



**Figure 6:** Selections of nodes along an anonymous tunnel

the selection  $\{3, 5, 55, 26, 31\}$  during the setup of the nested encryption with node 3. Note that node 1 is not interested in the selection it has presented itself and that there is no selection from the final node in the anonymous tunnel. Consequently, if there are  $n$  nodes in an anonymous tunnel, then the initiating node gets  $n - 2$  selections during the setup. After having set up the anonymous tunnel, node 1 knows the following according to figure 6: node 2 has offered the selection  $\{3, 5, 55, 26, 31\}$ , node 3 has offered the selection  $\{4, 48, 13, 17, 7\}$ , and node 4 has offered the selection  $\{5, 13, 36, 14, 1\}$ .

Each node maintains an internal table that contains one entry 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. We name this combination of node and selection the *extended selection*. The nodes within an extended selections are ordered in ascending order as their position within the selection is no longer relevant. Table 1 depicts the extended selections node 1 inserted into its table after having set up the anonymous tunnel according to figure 6.

**Table 1:** Extended selections after 1 anonymous tunnel

entry	extended selection
1	2,3,5,26,31,55
2	3,4,7,13,17,48
3	1,4,5,13,14,36

If the table contains only a few entries, then node 1 cannot say much about potentially colluding nodes. However, let's assume that node 1 has already set up several anonymous tunnels and its internal table looks as in table 2.

**Table 2:** Extended selections after 5 anonymous tunnels

entry	extended selection	entry	extended selection	entry	extended selection
1	2,3,5,26,31,55	4	2,14,23,33,41,47	7	11,12,24,29,43,53
2	3,4,7,13,17,48	5	11,16,33,37,42,52	8	8,14,35,43,46,50
3	1,4,5,13,14,36	6	7,11,16,22,33,42	9	6,7,29,31,35,49

node	extended selection	node	extended selection
10	2,15,28,34,37,55	13	3,5,14,17,47,51
11	11,16,26,33,37,42	14	11,12,19,32,47,52
12	7,22,26,37,42,52	15	16,22,33,37,42,52

We will now describe the computations a node performs to determine if an anonymous tunnel is composed of colluding nodes or not. Assume node 1 sets up a 6<sup>th</sup> anonymous tunnel and picks node 34 as the next hop. Node 34 delivers the selection  $\{9, 43, 12, 3, 48\}$ , which results in an extended selection of  $\{3, 9, 12, 34, 43, 48\}$ . Node 1 then computes the *correlation* of this extended selection 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 the beginning.
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, then 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  $s$ .
6. Compute the correlation  $c$  which is defined as  $c = m/s$  if  $s > 0$ , or 0 otherwise.

If we carry out this algorithm with the selection node1 has just received from node 34, the following is done:

1.  $ES_N = \{3, 9, 12, 34, 43, 48\}$
2.  $ES_R = \{\}$
3. Comparing  $ES_N$  with all lines in table 2, we see that entries 1, 2, 7, 8, 10, 13 and 14 contain at least one element of  $ES_N$ . This results in  $ES_R = \{\mathbf{2}, \mathbf{2}, \mathbf{3}, \mathbf{3}, \mathbf{3}, 4, \mathbf{5}, \mathbf{5}, 7, 8, \mathbf{11}, \mathbf{11}, \mathbf{12}, \mathbf{12}, 13, \mathbf{14}, \mathbf{14}, 15, \mathbf{17}, \mathbf{17}, 19, 24, 26, 28, 29, 31, 32, 34, 35, 37, \mathbf{43}, \mathbf{43}, 46, \mathbf{47}, \mathbf{47}, 48, 50, \mathbf{52}, \mathbf{52}, 53, \mathbf{55}, \mathbf{55}\}$
4.  $m = 23$  (the elements from  $ES_R$  typed in bold)
5.  $s = 19$
6.  $c = 23/19 = 1.21$

Assume the second selection of this anonymous tunnel comes from node 9 and is  $\{28, 46, 51, 8, 2\}$ , which results in

$$\begin{aligned}
 ES_N &= \{2, 8, 9, 28, 46, 51\} \\
 ES_R &= \{\mathbf{2}, \mathbf{2}, \mathbf{2}, \mathbf{3}, \mathbf{3}, \mathbf{5}, \mathbf{5}, 8, \mathbf{14}, \mathbf{14}, \mathbf{14}, 15, 17, 23, 26, 28, 31, 33, 34, 35, 37, 41, 43, 46, \mathbf{47}, \mathbf{47}, \\
 &\quad 50, 51, \mathbf{55}, \mathbf{55}\} \\
 m &= 14 \\
 s &= 16 \\
 c &= 14/16 = 0.88.
 \end{aligned}$$

Furthermore, the 4<sup>th</sup> node is 28 and offers the selection  $\{44, 54, 6, 12, 25\}$ , which gives

$$\begin{aligned}
ES_N &= \{6, 12, 25, 28, 44, 54\} \\
ES_R &= \{2, 6, 7, \mathbf{11}, \mathbf{11}, \mathbf{12}, \mathbf{12}, 15, 19, 24, 28, \mathbf{29}, \mathbf{29}, 31, 32, 34, 35, 37, 43, 47, 49, 52, 53, 55\} \\
m &= 6 \\
s &= 18 \\
c &= 6/18 = 0.33.
\end{aligned}$$

What we have seen is that after carrying out the algorithm, the three extended selections result in the correlations 1.21, 0.88, and 0.33. This alone does not tell much, but assume that the second selection of this anonymous tunnel is  $\{7, 28, 46, 14, 2\}$  instead of  $\{28, 46, 51, 8, 2\}$ , and the witness has selected node 7 instead of node 28 as the next hop. Consequently, it is node 7 that offers the third selection during the setup of the anonymous tunnel, which we assume to be  $\{42, 11, 26, 52, 33\}$ . When node 1 carries out algorithm 1, it gets

$$\begin{aligned}
ES_N &= \{7, 11, 26, 33, 42, 52\} \\
ES_R &= \{\mathbf{2}, \mathbf{2}, \mathbf{3}, \mathbf{3}, 4, 5, 6, \mathbf{7}, \mathbf{7}, \mathbf{7}, \mathbf{7}, \mathbf{11}, \mathbf{11}, \mathbf{11}, \mathbf{11}, \mathbf{11}, \mathbf{12}, \mathbf{12}, 13, 14, \mathbf{16}, \mathbf{16}, \mathbf{16}, \mathbf{16}, 17, 19, \mathbf{22}, \\
&\quad \mathbf{22}, \mathbf{22}, 23, 24, \mathbf{26}, \mathbf{26}, \mathbf{26}, \mathbf{29}, \mathbf{29}, \mathbf{31}, \mathbf{31}, 32, \mathbf{33}, \mathbf{33}, \mathbf{33}, \mathbf{33}, \mathbf{33}, 35, \mathbf{37}, \mathbf{37}, \mathbf{37}, \mathbf{37}, 41, \\
&\quad \mathbf{42}, \mathbf{42}, \mathbf{42}, \mathbf{42}, \mathbf{42}, 43, \mathbf{47}, \mathbf{47}, 48, 49, \mathbf{52}, \mathbf{52}, \mathbf{52}, \mathbf{52}, 53, 55\} \\
m &= 49 \\
s &= 17 \\
c &= 49/17 = 2.88.
\end{aligned}$$

The resulting value 2.88 is much bigger than the one in the three previous cases. The reason for this is that in our example, the nodes 7, 11, 16, 22, 26, 33, 37, 42, and 52 collude. Consequently, whenever one of these nodes offers a selection, the extended selection contains only nodes from the colluding set, as can be seen in table 2 at entries 5, 6, 11, 12, and 15.

We will now explain why the resulting value 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 at the end of section 3.9.3 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 the algorithm, 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  $ES_R$ . For reasons (1–4), we can state the following properties about the set  $ES_R$ :

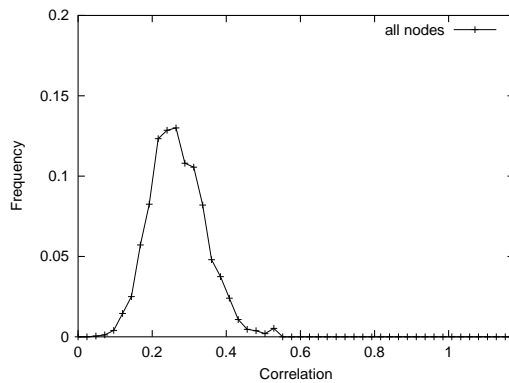
1. If the new extended selection  $ES_N$  mainly consists of colluding nodes,  $ES_R$  will contain relatively few different nodes and many occurrences of several colluding nodes, due to reasons (1) and (2). This implies a big  $m$  and a small  $s$ , resulting in a big  $c$ .
2. If the new extended selection  $ES_N$  mainly consists of well-behaving nodes,  $ES_R$  will contain relatively many different nodes with only a few of them occurring several times. This implies a small  $m$  and a big  $s$ , resulting in a small  $c$ .

One can argue why not simply counting the occurrences of elements in the new extended selection  $ES_N$  in the internal table. This would work if we assumed that every node in the system was equally popular, and 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, it is likely that some nodes are much more popular than others because of their spare bandwidth and computing power. When only counting 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 popular nodes from the colluding nodes is that although the popular nodes appear frequently in selections from 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 when there are some very popular nodes. Similarly, it would not be sufficient to look only at  $m$  instead of the ratio  $m/s$ . 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  $s$  into account:  $s$  tends to get relatively big compared to  $m$  when the new extended selection contains mainly well-behaving nodes – independently of the popularity of the nodes, but is relatively small compared to  $m$  when the extended selection consists of malicious nodes.

#### 4.1 Detecting Malicious Tunnels

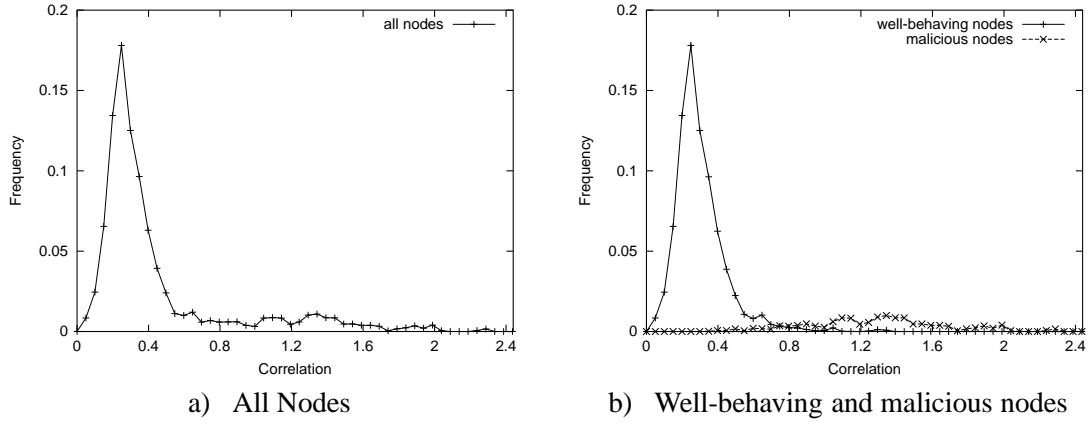
We have seen that high correlations are an indication for colluding nodes. However, we have not given a limit above which an extended selection gets suspicious. The problem is that there is no such fixed limit. The correlations depend on the number of different nodes, their popularity, the number of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

A node remembers the correlations it has computed over time and stores them. They can be represented as a distribution function, as we will show below. The goal is to deduce a limit for the correlation to identify potentially malicious tunnels. We give some examples of these distribution functions below. We start with a system consisting of 1'000 nodes. Each node is equally popular. We set up 5'000 anonymous tunnels, whereas each tunnel consists of 5 hops in total, which means that we get 3 selections during each setup – one from each of the intermediate nodes – and 15'000 in total. The other parameters are varied. Figure 7 shows the correlation distribution when there are no malicious nodes.



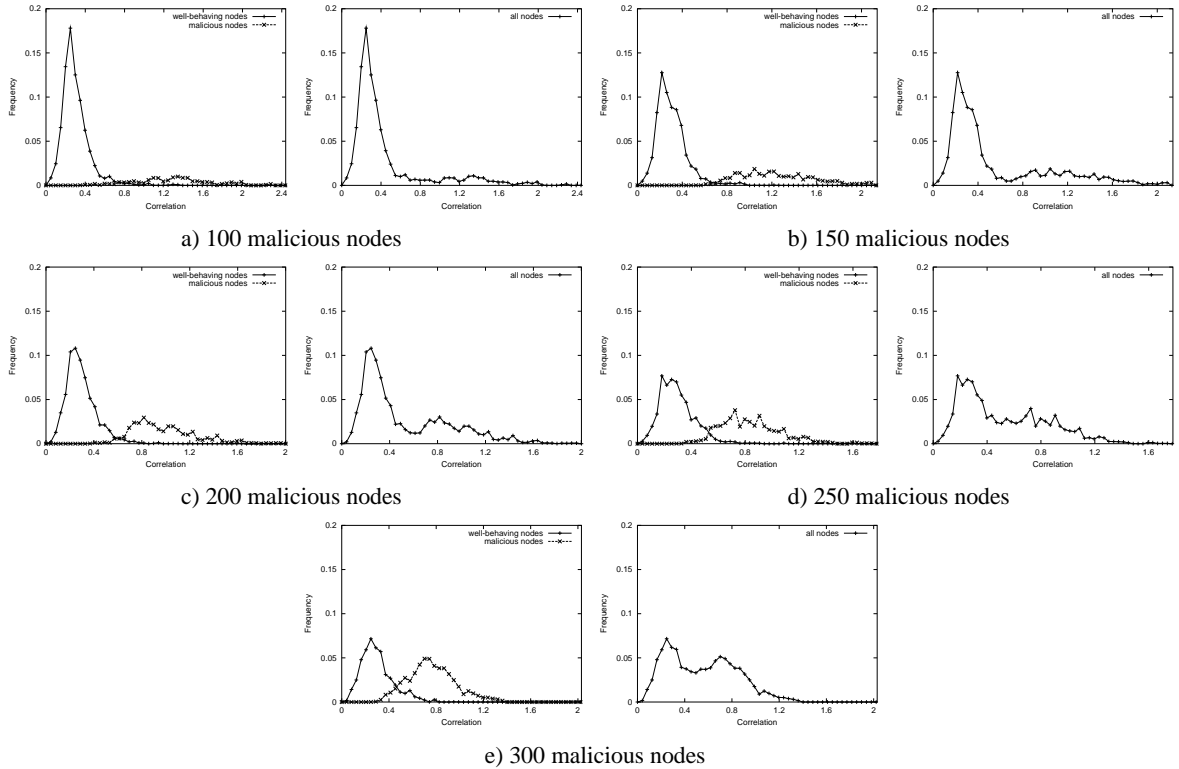
**Figure 7:** *Correlation distribution with only well-behaving nodes*

We see that all correlations are centered around 0.2. While this is not very interesting yet as there are no malicious nodes, it tells us nevertheless that all correlations of well-behaving nodes seem to be in the same range. In figure 8, we look at the correlation distribution when 50 of the 1'000 nodes are malicious and build the colluding set.



**Figure 8:** Correlation distribution with 50 malicious nodes

Figure 8 a) shows the distribution of all nodes, whereas b) illustrates the distributions for well-behaving and malicious nodes separately. While the initiator of course gets only the distribution of all nodes, the separate distributions are interesting to look at to evaluate if our approach works. Indeed, looking at figure 8 b), we see that nearly all of the well-behaving nodes are below 0.4 and nearly all malicious nodes are above that value. This again supports our approach based on the assumption that a high correlation for a new extended selection is always an indication that the node offering the selection may be malicious. Figure 9 shows the same results when the colluding set consists of 100, 150, 200, 250, and 300 nodes.



**Figure 9:** Correlation distribution with 100, 150, 200, 250, and 300 colluding nodes

Figures 9 a)–e) illustrate what happens when the number of colluding nodes gets larger: (1) the number of extended selections received from malicious nodes grows and gets more and more visible as

a second peak in the correlation distribution, and (2) the overlapping range of the well-behaving and malicious nodes gets larger. This overlapping range is hardly visible in figure 9 a), but is significant in figure 9 e) where it reaches from 0.22 to 0.44.

So what is the strategy a node follows when trying to detect anonymous tunnels consisting of malicious nodes? At any time, the node knows the correlation distribution it has generated based on selections it received previously during the setup of other anonymous tunnels. Based on this distribution, the node determines a *correlation limit*. This limit correlation should have the property that if the correlation of an 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*. Note that if only the final node in the tunnel is malicious, then this is difficult to detect as it does not offer a selection. However, this final node cannot learn anything about the anonymous tunnel by itself. On the other hand, if at least one extended selection is above the limit, the tunnel is considered *malicious* and will not be used. Looking at figure 8, a limit of about 0.35 would be best in that particular situation. The difficulty of determining this limit lies in the fact that the node only knows the correlation distribution of all nodes, i.e. the information illustrated in figure 7, figure 8 a), and the left sides of figures 9 a)–e). Finding a good correlation limit from these distributions is not easy, especially if the overlapping range of the well-behaving and malicious nodes is large.

Algorithm 2 lists the steps the initiator carries out during the setup of an anonymous tunnel to determine whether it is considered good or malicious.

**Algorithm 2** *Determining if an anonymous tunnel is good or malicious*

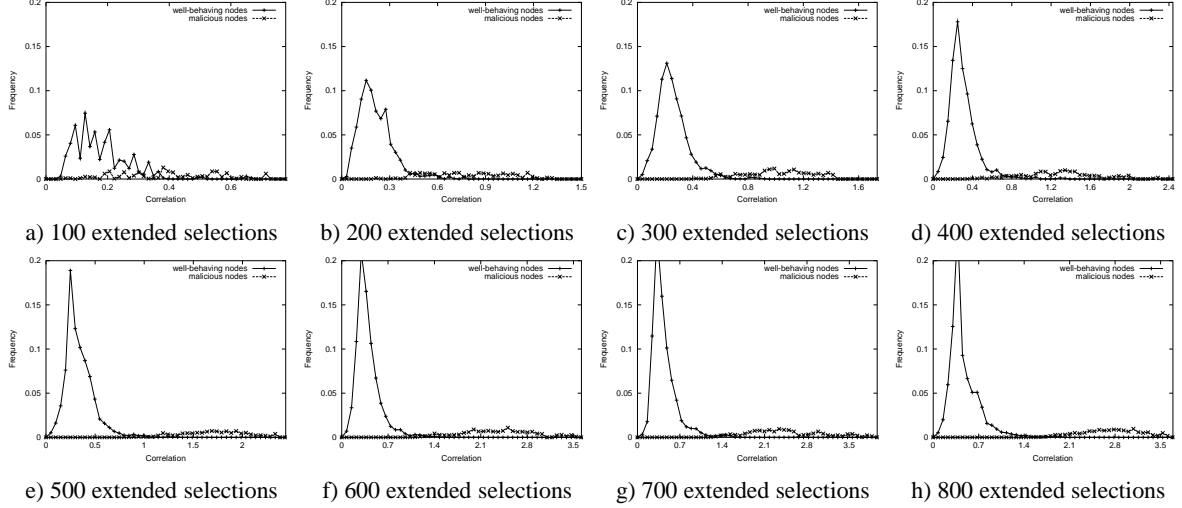
1. Initialize a variable *rejectTunnel* to false.
2. Get the next extended selection  $ES_N$  of the anonymous tunnel.
3. Compute the correlation  $c$  of  $ES_N$  according to algorithm 1.
4. Determine the limit correlation  $c_l$  from the correlation distribution.
5. If  $c$  is greater than  $c_l$ , 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.

## 4.2 Size of the Internal Table

Computing the correlation of an extended selection has complexity proportional to the number of extended selections in the internal table. We should therefore try to keep the size of the internal table 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. With a *received extended selection* we mean the combination of a received selection and the node that offered that selection. This is not only reasonable to keep the complexity low, but also makes sense as new extended selections give the most accurate picture of the current situation of the system. As nodes may join and leave the system

at any time, old extended selections could contain several nodes that have left the system a while ago.

To find out a reasonable size for  $k$ , we analyze how the correlation distribution looks depending on the number of extended selections in the internal table. We use the same setting as in section 4.1 and set the number of colluding nodes to 100. Figure 11 depicts the correlation distributions of well-behaving and malicious nodes when the size of the internal table varies from 100 to 800 extended selections.



**Figure 10:** *Correlation Distribution depending on the Size of the Internal Table*

Figure 10 shows that above a certain number of extended selections in the internal table, making the size of the table bigger does not significantly improve the quality of the resulting correlation distribution. In figure 10, the quality improves from a) to d), but from d) to h), there is no significant difference. We conclude that in our example, a reasonable lower limit for the number of extended selections in the internal table is 400. Note that this does not mean that 400 extended selections is always the right choice when there are 1'000 nodes. In our example, each selection contains 5 nodes, which means the 400 extended selections correspond to 2400 nodes. We can expect that more extended selections are needed in the internal table if there are only 3 nodes in a selection. Indeed, experiments show that about 650 extended selections are needed in that case.

We have performed many experiments and have derived a formula for a reasonable size for the internal table. This size depends on the number of nodes in the system, but this is not known a priori in a real life environment. All a node can do is counting the different nodes it has seen so far, which is perfectly all right since this number tells the node the size of that part of the system it has already had contact with. If  $n$  is the number of different nodes that the node has seen so far and  $\bar{s}$  is the average number of elements in a selection, then the number of extended selection  $k$  in the internal table should be

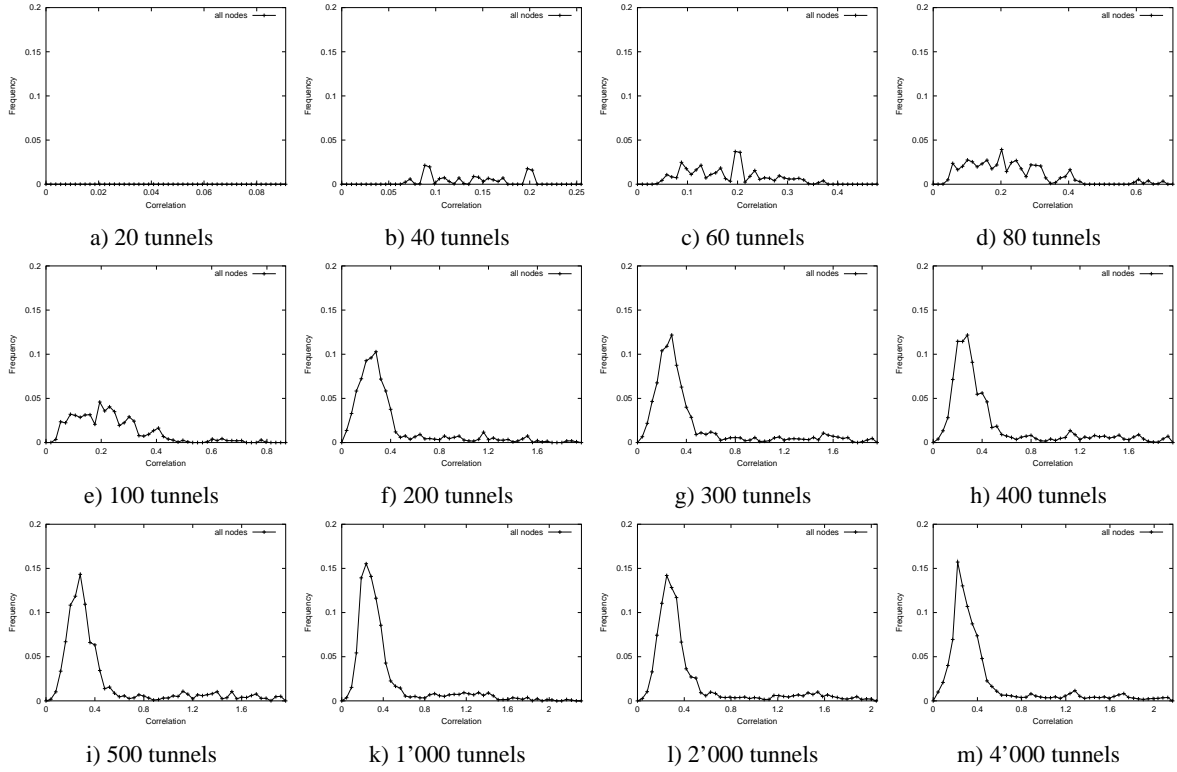
$$k = 2 \cdot \frac{n}{\bar{s}}. \quad (1)$$

Using the example above and assuming that after 5'000 anonymous tunnels, the node has seen all 1'000 nodes in the system, we get  $k = 2 \cdot 1'000/5 = 400$  extended selections, which corresponds to figure 10 d).

### 4.3 Behavior and Implementation of the Correlation Distribution

Another question is how the correlation distribution develops itself starting with an empty internal table. We use the same setting as in section 4.1, set the number of colluding nodes to 100, and the size of the

internal table to 400 extended selections nodes (according to formula 1). Figure 11 depicts the correlation distributions of all nodes after having set up 20–4'000 anonymous tunnels.



**Figure 11:** *Correlation distribution depending on the number of anonymous tunnels*

Figure 11 gives a lot of information about the correlation distribution. First of all, when starting with an empty internal table, the correlation distribution builds up slowly and only after having set up several anonymous tunnels the distribution is getting its shape. With 1'000 nodes, figure 11 b) and c) show that after about 40–60 tunnels, the distribution starts building its typical peak. This is not surprising as correlations  $c > 0$  of new extended selections only show up if the internal table contains at least one extended selection that has a node in common with the new extended selection (see steps 3, 5, and 6 of Algorithm 1). But this also implies that making a reasonable decision whether a node offering a selection is malicious or not is only possible after the internal table has been filled with some extended selections, i.e. after the correlation distribution starts getting some shape. We will come back to this discussion in section 4.5.

A second property visible from figure 11 is that the correlation distribution builds up steadily until it has reached a certain shape. After that it does not change a lot. In figure 11, the distribution has reached its shape after about 400 tunnels. From figures 11 h) to m) – despite minor variations – the basic form of the distribution remains the same.

The correlation distribution has additional properties that are not directly visible from figure 11. It is implemented as an array *dist*, whereas each slot of the array corresponds to a particular correlation,  $corr[i]$ . For example,  $corr[0] = 0, corr[1] = 0.1, corr[2] = 0.2, corr[3] = 0.3, corr[4] = 0.4 \dots$ . Whenever we compute a new correlation, we add a total value of 1 to the slots of *dist*. If a new correlation  $c = 0.3$  is computed with algorithm 1, we add 1 to  $dist[3]$ , since  $corr[3] = 0.3$ . If a new correlation  $c = 0.26$  is computed, we add 0.4 to  $dist[2]$  and 0.6  $dist[3]$ . To all other slots of *dist*, we add nothing. We represent these values we add to *dist* after having computed a new correlation as an array *new\_values* that has the same number of slots as *dist*. Note that all except one or two slots of *new\_values* are always zero.

Correlations of new extended selections should affect the distribution correlation more than those of old extended selections. The reason is the same as during the discussion of the size of the internal table in section 4.2: new extended selections give a more accurate picture of the current state of the system. Consequently, when a new correlation has been computed, we do not simply increment the values of the corresponding slots of the array, as this would mean that new correlations would not affect the distribution more than those computed a long time ago. Rather, the value of each slot follows an exponential weighted moving average (EWMA) with parameter  $\alpha$ .  $dist[i] = \alpha \cdot dist[i] + (1 - \alpha) \cdot new\_values[i]$ . Assume that the array representing the discrete distribution is named *dist* and *new\_values* is the array representing the new values that should be added to the slots  $i$  of *dist*. For each newly computed correlation and each slot  $i$  of the array, the following is applied:

$$dist[i] = \alpha \cdot dist[i] + (1 - \alpha) \cdot new\_values[i] \quad (2)$$

$\alpha$  itself is not fixed but depends on the number of extended selections in the internal table. If  $t$  is the number of extended selections in the internal table, then:

$$\alpha = \begin{cases} 0 & \text{if } t = 0 \\ 1 - \frac{1}{t} & \text{otherwise} \end{cases} \quad (3)$$

This means that the more extended selections there are in the internal table, the closer  $\alpha$  is to 1, i.e. the slower the influence of older extended selections diminishes. If there are 100 extended selections in the internal table, then  $\alpha = 0.01$ , which means that the current distribution is based 1% on the most recent correlation, 0.99% on the second most recent correlation, 0.9801% on the third most recent correlation and so on.

Starting with an empty correlation distribution and adding a total of 1 to the slots of *dist* with each newly computed correlation implies that the sum of all  $dist[i]$  approaches 1 as the internal table gets more and more filled. Corresponding to figure 10, table 3 shows the sum of all  $dist[i]$  depending on the number of anonymous tunnels.

**Table 3: Sum of all  $dist[i]$**

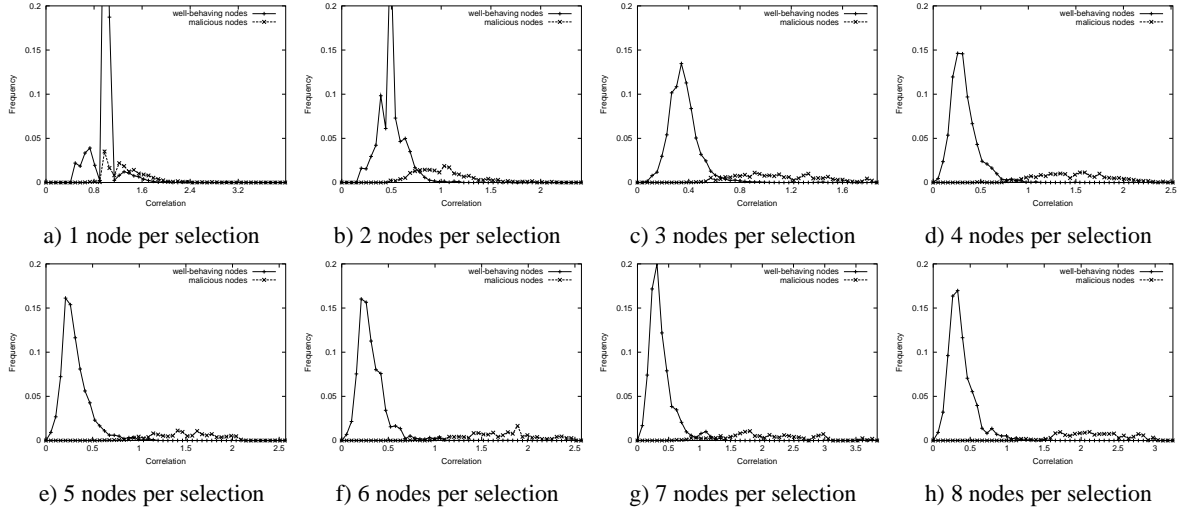
Anonymous Tunnels	sum( $dist[i]$ )	Anonymous Tunnels	sum( $dist[i]$ )
20	0.0	300	0.903
40	0.042	400	0.954
60	0.257	500	0.976
80	0.414	1'000	0.998
100	0.525	2'000	0.999
200	0.790	4'000	0.999

The sum starts from 0 and gets closer to 1 as more and more anonymous tunnels are set up. Note that after a while, it remains somewhere between 0.98 and 0.99 but never gets closer to 1. The reason for this is that occasionally, the sum of all elements in *new\_values* is smaller than 1 because there are some rare cases where the computed correlation does not lie within the current range of *dist*.

Currently, *dist* consist of 50 slots as our experiments have shown this to be a good enough discrete representation of the real continuous correlation distribution. The correlation values corresponding to the slots are not fixed. Rather, they are adjusted if needed such that the range of the array covers nearly all correlations computed by algorithm 1. This is why the range of the x-axis on figure 11 is changing, especially until the correlation distribution has reached a certain shape.

#### 4.4 The Role of the Selection Size

Until now, we have always used 5 as the number of nodes in a selection. In this section, we want to analyze how the correlation distribution depends on the selection size. Using the same setting as in the previous section, figure 12 shows the correlation distributions of well-behaving and malicious nodes when the selections contains 1–8 nodes.



**Figure 12:** *Correlation Distribution depending on the Selection Size*

We see that the overlapping range of the well-behaving and malicious nodes is quite large when the selection size is only 1 or 2 nodes. Using 3 nodes gives already quite a small overlapping range and with a selection size of 5 we have nearly separated the distributions of well-behaving and malicious nodes. Furthermore, increasing the selection size beyond 5 does not seem to give better results. We conclude that with 1'000 nodes in the system, the selection size should not be smaller than 3 and 5 seems to be a reasonable choice. There is no reason to use a selection size greater than 5.

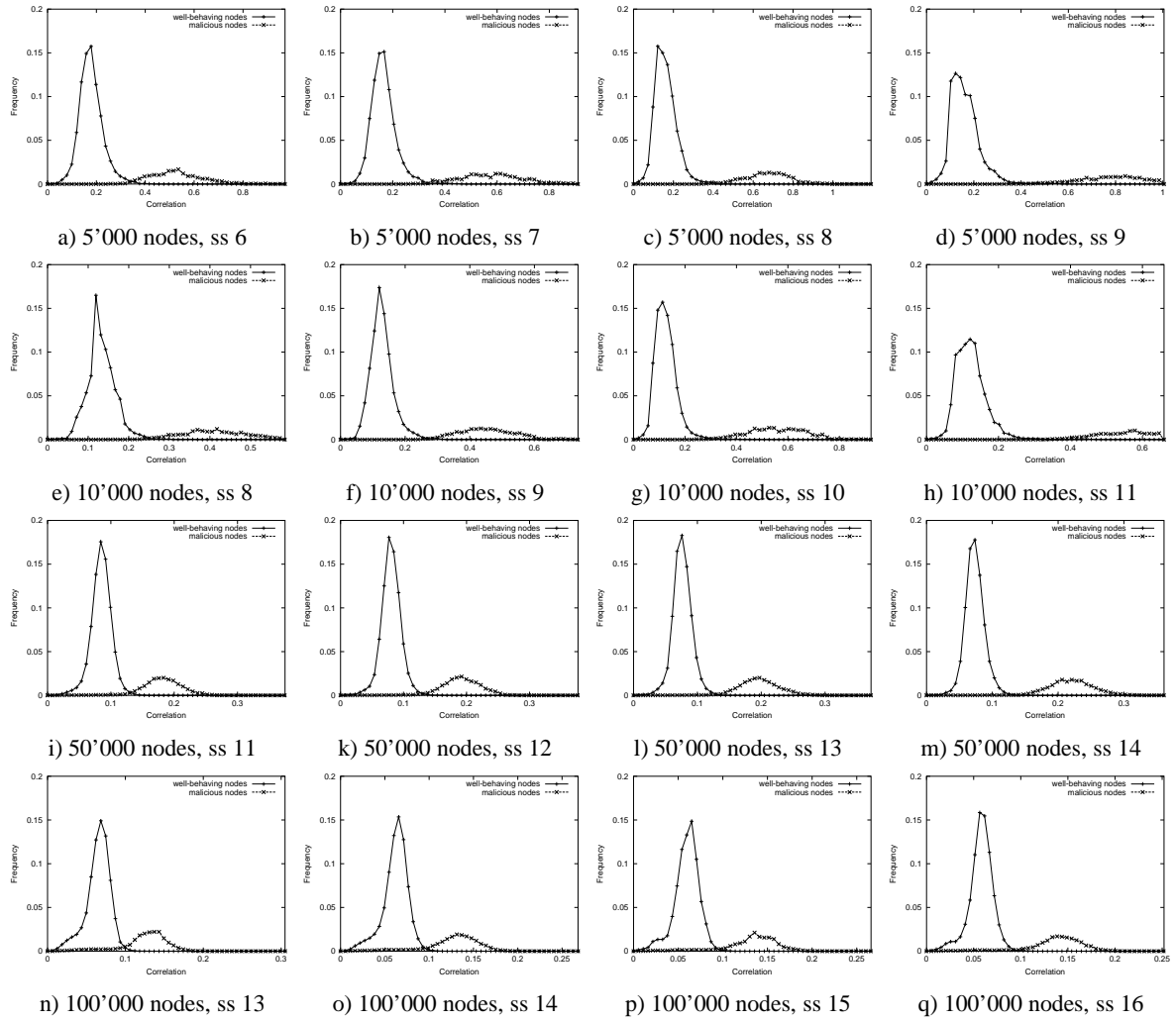
The question is if this reasonable selection size is dependent on the number of nodes in the system or not. We have analyzed this for systems consisting of 5'000–100'000 nodes. We always set up about as many tunnels as there are nodes in the system. The number of colluding nodes is always 10% of all nodes. Each node is equally popular. Figure 13 illustrates the results.

Figure 13 shows a particular range of the selection size (ss in figure 13). We see that the reasonable selection size depends on the number of nodes in the system. While 5 nodes in each selection when there are 1'000 nodes in the system is good enough to get a small overlapping range between well-behaving and malicious nodes (figure 12 d), we need about 8 nodes per selection with 5'000 nodes in the system, about 10 with 10'000, about 13 with 50'000, and a selection size of about 15 when the system consists of 100'000 nodes.

From our experiments, we have derived a formula for a reasonable selection size. This size depends on the number of nodes in the system, but again, this is not known a priori in a real life environment. Therefore, the initiating node again counts the different nodes it has seen so far. If  $n$  is the number of different nodes that the node has seen so far, then the number of nodes  $s$  in a selection should be

$$s = \begin{cases} \lceil 5 \cdot \log_{10} d - 10 \rceil & \text{if } n > 158 \\ 1 & \text{otherwise} \end{cases} \quad (4)$$

Replacing  $n$  in formula 4 with values ranging from 1 to 1'000'000, we get the nodes range – selection size pairs according to table 4.



**Figure 13:** Correlation distributions depending on the total number of nodes and the selection size

**Table 4:** Selection size depending on the number of total nodes

different nodes seen	proposed number of nodes in selection	different nodes seen	proposed number of nodes in selection
1–158	1	10'001–15'848	11
159–251	2	15'849–25'118	12
252–398	3	25'119–39'810	13
398–630	4	39'811–63'095	14
631–1'000	5	63'096–100'000	15
1'001–1'584	6	100'001–158'489	16
1'584–2'511	7	158'490–251'188	17
2'512–3'981	8	251'189–398'107	18
3'982–6'309	9	398'108–630'957	19
6'310–10'000	10	630'958–1'000'000	20

Comparing the values for the selection size in table 4 with figure 13, we see that the formula indeed delivers reasonable values for the number of nodes in a selection. In general, the selection size according to formula 4 is usually a little more than the reasonable selection size. For example, we have discussed

above that according to figure 13 e)–h), a selection size of 10 seems to be appropriate when there are 10'000 nodes in the system. However, according to table 4, the selection size is increased to 11 as soon as the number of different nodes goes beyond 10'000. This guarantees that once the number of nodes gets so high that a selection size of 11 is indeed appropriate, there will already be several extended selections corresponding to that size in the internal table.

From now on, we will not anymore specify a fixed value (e.g. 5) for the selection size as we did up to now, but use formula 4 to compute the selection size depending on the number of different nodes the initiating node has seen at any moment.

## 4.5 A First Performance Evaluation

We now analyze how well our algorithm performs depending on the number of nodes in the system. There are basically two figures we are evaluating: *false positives*, i.e. the number of good anonymous tunnels that algorithm 2 has wrongly classified as malicious, and the *false negatives*, which are those anonymous tunnel that have been classified as good but actually contain more than one malicious node. According to our assumption stated in section 3.9.3, we assume 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  $1 \dots (n - 1)$  malicious nodes. As we do not get a selection from the final node in the tunnel, we cannot detect anonymous tunnels where only this final node is malicious. However, this is not a problem as the final node itself cannot learn anything about the anonymous tunnel by itself. Consequently, we try to detect tunnels consisting of  $2 \dots (n - 1)$  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.

Up to now, we have always assumed that each node is equally popular, which means that if a node picks other nodes to be used in a selection, then each of the other nodes is picked with the same probability. In reality, this is hardly the case as some nodes will always be more popular than others. From now on, we model the different popularities of the nodes with a negative exponential distribution such that the most popular nodes are about 50 times as popular as the least popular nodes.

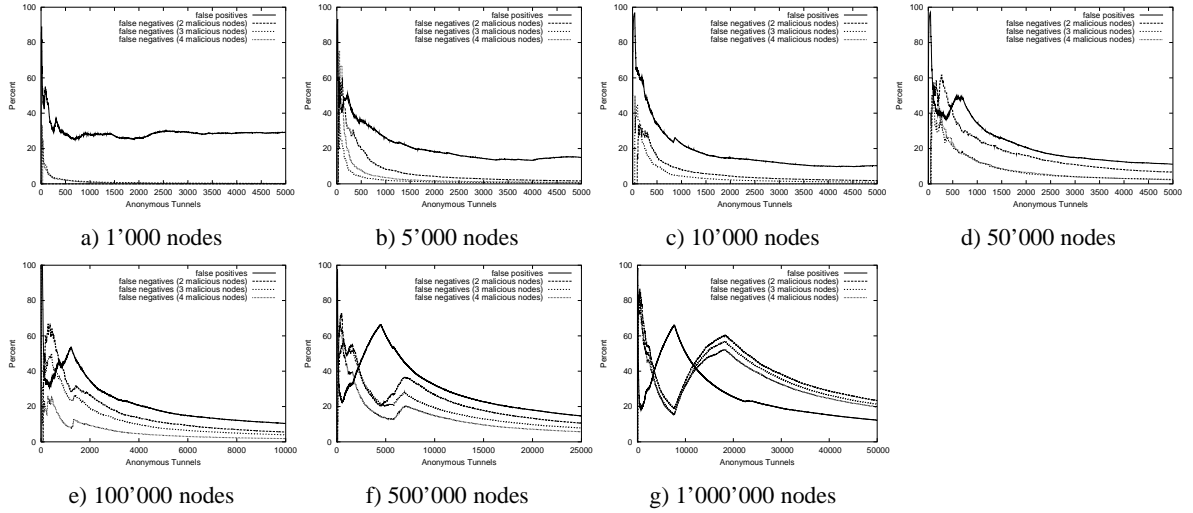
Figure 14 shows the false positives and negatives when using systems with 1'000–1'000'000 nodes. The other parameters are the same as in section 4.4. The selection size is determined according to formula 4. Each figure shows the percentage of false positives and negatives after  $n$  anonymous tunnels have been set up. For instance, figure 14 d) shows that the line of the false positives is about at 20% after 2'500 anonymous tunnels. This means that there have been 20% false positives during the first 3000 anonymous tunnels.

We can see in figure 14 that the percentage of false positives and false negatives are getting smaller as the number of anonymous tunnels grows, which is good. However, we can also see that when only relatively few tunnels have been set up, it is difficult to clearly determine if a new extended selection is good or bad. It seems that it takes about 50 tunnels with 1'000 nodes in the system, about 200 tunnels with 10'000 nodes, about 1'000 tunnels with 100'000 nodes, and about 20'000 tunnels with 1'000'000 nodes until the lines in figure 14 are clearly decreasing, i.e. until algorithm 2 delivers the correct result in most cases. Recalling the discussion following figure 11 in section 4.3, this is not surprising as it takes some time until the correlation distribution starts building its typical peak. The more different nodes there are in the system, the longer this takes.

Table 5 shows the precise numbers of the false positives and negatives after having set up all anonymous tunnels according to figure 14 a–g.

We can see that up to about 100'000 nodes, there are only a few false negatives we have missed. With 500'000 nodes and above however, the relatively long time it takes to collect enough anonymous tunnels to make meaningful statements about the goodness or badness of an anonymous tunnel results in several malicious tunnels that remain undetected.

The question remains if we can reduce the number of anonymous tunnels we need to set up before



**Figure 14:** Performance when adapting the selection size according to formula 4

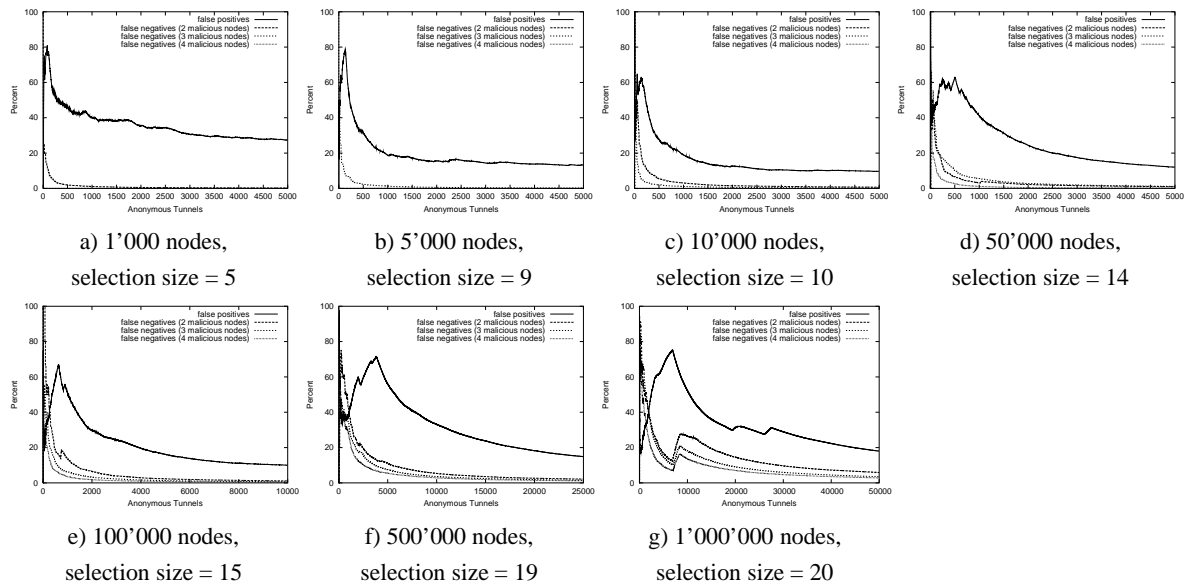
**Table 5:** False positives and negatives (figure 14)

	1'000 nodes	5'000 nodes
false positives	1'058 of 3'622 → 29.21%	539 of 3'614 → 14.91%
false negatives (2 malicious nodes)	0 of 424 → 0.00%	7 of 425 → 1.65%
false negatives (3 malicious nodes)	1 of 442 → 0.23%	2 of 448 → 0.45%
false negatives (4 malicious nodes)	1 of 512 → 0.20%	4 of 513 → 0.78%
	10'000 nodes	50'000 nodes
false positives	373 of 3'650 → 10.22%	413 of 3'728 → 11.08%
false negatives (2 malicious nodes)	7 of 382 → 1.83%	27 of 404 → 6.68%
false negatives (3 malicious nodes)	4 of 442 → 0.90%	10 of 393 → 2.54%
false negatives (4 malicious nodes)	0 of 526 → 0.00%	11 of 475 → 2.32%
	100'000 nodes	500'000 nodes
false positives	761 of 7'315 → 10.40%	2'657 of 18'111 → 14.67%
false negatives (2 malicious nodes)	45 of 812 → 5.54%	215 of 2'027 → 10.61%
false negatives (3 malicious nodes)	35 of 887 → 3.95%	175 of 2'234 → 7.83%
false negatives (4 malicious nodes)	18 of 986 → 1.83%	149 of 2'628 → 5.67%
	1'000'000 nodes	
false positives	4'460 of 36'282 → 12.29%	
false negatives (2 malicious nodes)	952 of 4'071 → 23.38%	
false negatives (3 malicious nodes)	956 of 4'485 → 21.32%	
false negatives (4 malicious nodes)	1'018 of 5'162 → 19.72%	

the correlation distribution gets useful. We have mentioned in section 4.3 that useful values for the correlation of new extended selections only show up after the internal table contains at least one extended selection that has a node in common with the new extended selection. Consequently, it can be expected that the correlation distribution gets useful faster if the extended selections contain many nodes. Assume that the initiating node knows beforehand how many nodes there are in the system. It would then be reasonable to choose the selection size corresponding to the number of nodes in the system from the beginning and not adopt the selection size depending on the number of different nodes the initiator has observed.

In figure 15, we analyze the performance again, but this time we assume the initiator knows the

number of nodes in the system and uses the corresponding selection size from the beginning. According to formula 4, this ranges from a selection size of 5 with 1'000 nodes in the system to a selection size of 20 with 1'000'000 nodes in the system.



**Figure 15:** Performance choosing the selection size according to the number of nodes in the system

Table 6 shows again the precise numbers of the false positives and negatives after having set up all anonymous tunnels according to figure 15 a–g.

**Table 6:** False positives and negatives (Figure 15)

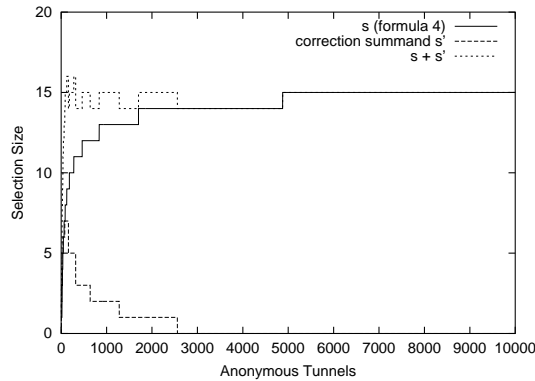
	1'000 nodes	5'000 nodes
false positives	992 of 3'629 → 27.34%	485 of 3'626 → 13.38%
false negatives (2 malicious nodes)	1 of 429 → 0.23%	0 of 409 → 0.00%
false negatives (3 malicious nodes)	0 of 460 → 0.00%	1 of 452 → 0.22%
false negatives (4 malicious nodes)	0 of 482 → 0.00%	0 of 513 → 0.00%
	10'000 nodes	50'000 nodes
false positives	344 of 3'562 → 9.68%	441 of 3'670 → 12.02%
false negatives (2 malicious nodes)	3 of 431 → 0.70%	4 of 408 → 0.98%
false negatives (3 malicious nodes)	1 of 446 → 0.22%	5 of 439 → 1.14%
false negatives (4 malicious nodes)	0 of 561 → 0.00%	1 of 483 → 0.21%
	100'000 nodes	500'000 nodes
false positives	725 of 7'265 → 9.98%	2'718 of 18'296 → 14.86%
false negatives (2 malicious nodes)	9 of 808 → 1.11%	43 of 2'005 → 2.14%
false negatives (3 malicious nodes)	6 of 922 → 0.65%	33 of 2'234 → 1.48%
false negatives (4 malicious nodes)	4 of 1'005 → 0.40%	28 of 2'465 → 1.14%
	1'000'000 nodes	
false positives	6'533 of 36'333 → 17.98%	
false negatives (2 malicious nodes)	239 of 4'071 → 5.87%	
false negatives (3 malicious nodes)	163 of 4'606 → 3.54%	
false negatives (4 malicious nodes)	141 of 4'990 → 2.83%	

Figure 15 shows that the number of anonymous tunnels that must be set up before the correlation

distribution gets useful could indeed be reduced. With 1'000 nodes, only a few anonymous tunnels have to be set up before the percentage of false starts decreasing. With 10'000 nodes, we get satisfactory results after about 100 tunnels, with 100'000 nodes, we must wait approximately 750 tunnels, and with 1'000'000 nodes, about 8'000 anonymous tunnels have to be set up before the percentages of both false positives and false negatives clearly decrease. To summarize, the number of anonymous tunnels that are needed before the correlation gets useful could be reduced by a factor of 3–4 when using the selection size corresponding to the total number of nodes in the system from the beginning.

The question is how to determine this total number of nodes when the initiating node has only set up a few anonymous tunnels and has consequently only seen a small subset of all nodes in the system? We have seen in section 4.3 that the sum of the discrete values of the correlation distribution starts from 0 and grows close to 1 as the correlation of new extended selections are computed. The more nodes there are in the system, the slower the sum grows to 1, i.e. the more extended selections have to be collected until the sum approaches 1. We make use of this property to estimate a reasonable selection size as early as possible. We still use formula 4 as the basis, but depending on the sum of the values of the correlation distribution, we add a summand  $s'$  to  $s$ . If the sum grows rapidly to 1, then  $s'$  is small or even 0. On the other hand,  $s$  can become quite big if the sum grows only slowly.

Figure 16 depicts the selection size chosen by the initiating node when using  $s$  according to formula 4 and when adding the correction summand  $s'$ . In this example, we use a system with 100'000 nodes whereas the other parameters follow the previous examples.



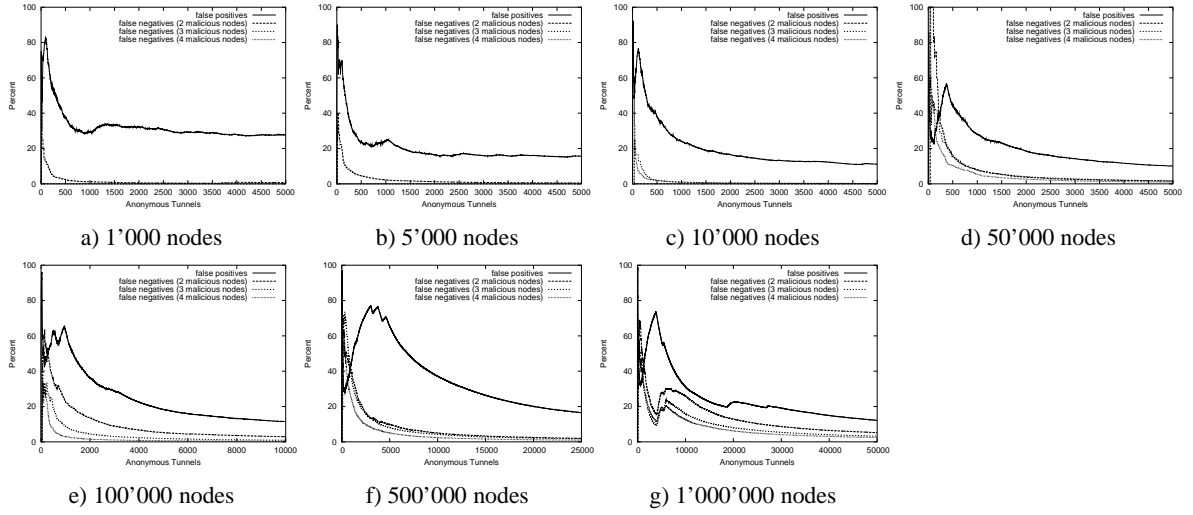
**Figure 16:** Selection size with and without correction summand  $s'$

One can easily see that the correction summand helps a lot. Without it, one gets close to the reasonable selection size of 15 only after having set up 1'700 anonymous tunnels. With the correction summand, the selection size is set close to 15 after about 80 tunnels already – at a time when  $s = 7$  according to formula 4. Note that the estimation is not completely precise. In the example above, the estimated reasonable selection size varies between 14 and 16 before the optimum of 15 is eventually found after about 4'900 tunnels. However, that is not a problem as according to figure 13, the correlation distribution does not change significantly if the selection size is a little above or below its reasonable size. In general, our estimation delivers a selection size which is quite close to the reasonable selection size. Figure 16 also shows the behavior of the correction summand  $s'$ : it goes up to 7 within the setup of 80 anonymous tunnels as the sum of the  $dist[i]$  does hardly grow at all in the beginning and slowly goes back to 0 as the sum starts growing more rapidly.

We expect that with the correction summand  $s'$ , we should come close to the results in figure 14 even without a priori knowledge of the total number of nodes in the system. Figure 17 shows the results.

Table 7 shows the precise numbers of the false positives and negatives after having set up all anonymous tunnels according to figure 17 a–g.

Comparing figure 17 with figure 15 we can see the results are about as good as when the initiator



**Figure 17:** Performance when using the correction summand  $s'$

**Table 7:** False positives and negatives (Figure 17)

	1'000 nodes	5'000 nodes
false positives	1'009 of 3'646 → 27.67%	570 of 1'344 → 15.59%
false negatives (2 malicious nodes)	3 of 417 → 0.72%	2 of 416 → 0.48%
false negatives (3 malicious nodes)	0 of 442 → 0.00%	0 of 431 → 0.00%
false negatives (4 malicious nodes)	0 of 495 → 0.00%	0 of 497 → 0.00%
	10'000 nodes	50'000 nodes
false positives	402 of 3'611 → 11.13%	367 of 3'641 → 10.08%
false negatives (2 malicious nodes)	0 of 411 → 0.00%	6 of 383 → 1.57%
false negatives (3 malicious nodes)	1 of 464 → 0.22%	8 of 467 → 1.71%
false negatives (4 malicious nodes)	1 of 514 → 0.19%	5 of 509 → 0.98%
	100'000 nodes	500'000 nodes
false positives	838 of 7'320 → 11.45%	3'005 of 18'214 → 16.50%
false negatives (2 malicious nodes)	25 of 866 → 2.89%	40 of 2'040 → 1.96%
false negatives (3 malicious nodes)	8 of 880 → 0.91%	38 of 2'239 → 1.70%
false negatives (4 malicious nodes)	3 of 934 → 0.32%	23 of 2'507 → 0.92%
	1'000'000 nodes	
false positives	4'425 of 36'434 → 12.15%	
false negatives (2 malicious nodes)	208 of 4'001 → 5.20%	
false negatives (3 malicious nodes)	146 of 4'554 → 3.21%	
false negatives (4 malicious nodes)	123 of 5'011 → 2.45%	

knows the number of nodes in the system a priori. This is not surprising, as figure 16 has demonstrated that the estimated selection size gets close to the reasonable selection size after having set up relatively few anonymous tunnels. As a conclusion, we will always use the correction summand  $s'$  when computing the selection size from now on. Consequently, formula 5 replaces formula 4.

$$s = \begin{cases} \lceil 5 \cdot \log_{10} d - 10 \rceil + s' & \text{if } n > 158 \\ 1 + s' & \text{otherwise} \end{cases} \quad (5)$$

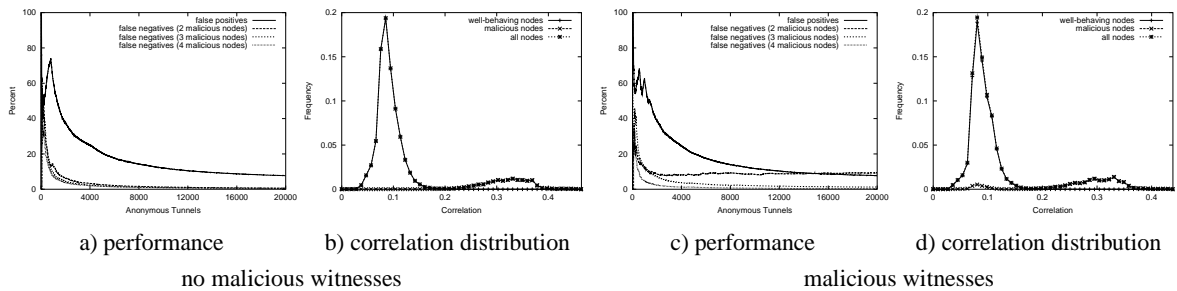
## 4.6 The Influence of Malicious Witnesses

So far, we have not taken into account the influence of malicious witnesses. If the witness that picks the next hop is in the same colluding set as the node that offers the selection, they can easily produce a fake selection for the initiating node, as we have discussed in section 3.9.3 (case 3). A clever adversary would simply generate a selection consisting of nodes that are not in the same colluding set as the witness and the node offering the selection. The node can then simply pick any next hop node from the same colluding set or even simulate the next hop by itself. The initiator of the tunnel cannot tell the difference: all it has received is a receipt signed by the witness containing the selection, which looks perfectly good.

Assume that  $B$  is the node that offers the selection,  $W$  is the witness for setting up the next hop, and  $B$  and  $W$  happen to be in the same colluding set.  $B$  and  $W_1$  build a fake selection  $\{O', P', Q', R', S'\}$  consisting of nodes that are not in the same colluding set as  $B$  and  $W$ . For the initiator  $A$ , this results in a new extended selection  $ES_N = \{B, P', Q', R', S', T'\}$ . In addition,  $A$  believes that  $P'$  is the next hop as it is the first node in the selection. In reality, however, it is  $B$  itself that also acts as the next hop. Now assume that for the setup of the following hop,  $A$  chooses  $W_2$  as the witness, and  $W_2$  does not happen to be in the same colluding set as  $B$  and  $W_1$ . In this case,  $B$  must offer a legitimate selection to  $W_2$ , and as  $B$  wants to have a node its own colluding set to be selected for the next hop, the selection also contains nodes from  $B$ 's colluding set.  $B$ 's selection could therefore be  $\{F, G, H, I, K\}$ , the nodes of which are all colluding with  $B$ . At  $A$ 's side, this results in a new extended selection  $ES_N = \{P', F, G, H, I, K\}$  since  $A$  still believes that the selection stems from  $P'$ . We conclude that if the node offering the selection and the witness for the setup of that hop are in the same colluding set, this results in 2 modified extended selections at  $A$ 's side. The first one ( $\{B, P', Q', R', S', T'\}$  above) contains only fake nodes ( $P' - T'$ ) except one ( $B$ ) and the second ( $\{P', F, G, H, I, K\}$  above) contains one fake node ( $P'$ ). It can be expected that  $A$  is not able to detect that the first extended selection is a malicious one, since it looks very much like a “normal” extended selection where the nodes are selected more or less randomly. The second extended selection is less troublesome as it contains only one fake node.

Note that if  $W_2$  were also in the same colluding set as  $B$ , then this would result in another extended selection that contains only fake nodes except one at  $A$ 's side. This continues until  $A$  picks a witness that is not colluding with  $B$ .

We compare the performance of our algorithm when malicious witnesses are taken into account and when not. Figure 18 shows the false positives and negatives when there are 100'000 nodes in the system and 10'000 of them are malicious and in the same colluding set. Each tunnel consists of 5 nodes and 20'000 anonymous tunnels are set up. The other parameters follow those of the previous examples.



**Figure 18:** False positives and negatives depending on the tunnel length

Table 8 shows the precise numbers of the false positives and negatives for figure 18 after having set up 20'000 anonymous tunnels.

Figure 18 shows that especially when there are only 2 malicious nodes in a tunnel, the inclusion of malicious witnesses significantly increases the number of false positives. According to table 8, it goes up from 1.12% to 10.47%. The reason for this increase stems from situations where the setup of the last hop involves a node and a witness that are colluding. In that case, the resulting extended

**Table 8:** *False positives and negatives (Figure 18)*

	no malicious witnesses	malicious witnesses
false positives	1'121 of 14'615 $\rightarrow$ 7.67%	1'112 of 14'487 $\rightarrow$ 7.68%
false negatives (2 malicious nodes)	10 of 1'628 $\rightarrow$ 0.61%	150 of 1'611 $\rightarrow$ 9.31%
false negatives (3 malicious nodes)	8 of 1'745 $\rightarrow$ 0.46%	23 of 1'831 $\rightarrow$ 1.26%
false negatives (4 malicious nodes)	9 of 2'012 $\rightarrow$ 0.45%	8 of 2'071 $\rightarrow$ 0.39%

selection contains mostly fake nodes which makes it virtually impossible for the initiating node to detect this tunnel as malicious. With 3 or 4 malicious nodes in a tunnel, the figures when taking into account the malicious witnesses are also worse than without malicious witnesses, but still very low. The reason here is that although it occasionally happens that the witness and the node are in the same colluding set, the probability that this is true twice or even three times during the setup of an anonymous tunnel is small. In our example, 10% of all nodes are malicious and in the same colluding set. We can therefore say that if the node setting up the next hop is malicious, then the probability that the witness is also malicious is 0.1. So we can expect about 10% more false positives when there are 2 malicious nodes in a tunnel, which seems to be the case when looking at table 8. Similarly, when the tunnel consists of 3 malicious nodes, there is a probability of  $2 \cdot 0.9 \cdot 0.1 = 0.18$  that there is one witness colluding with the corresponding node and a probability  $(0.1)^2 = 0.01$  that this happens twice in the same tunnel. We therefore expect an increase of a little more 1% for the false negatives because in 1% of the tunnels with 3 malicious nodes, the initiator does not receive a malicious extended selection and in 18% of all cases it receives only one except of two. Looking at table 8 again confirms this as the number of false positives goes up 1.23%. When there are 4 malicious nodes in the tunnel, the corresponding probabilities are  $3 \cdot (0.9)^2 \cdot 0.1 = 0.243$  that there is once a coalition between the witness and the corresponding node,  $3 \cdot 0.9 \cdot (0.1)^2 = 0.027$  that this happens twice in an anonymous tunnel, and  $(0.1)^3 = 0.001$  that this is the case three times. Consequently, we expect the percentage of false negatives to raise more than 0.1%, which is again confirmed as table 8 shows this increase to be 0.34%.

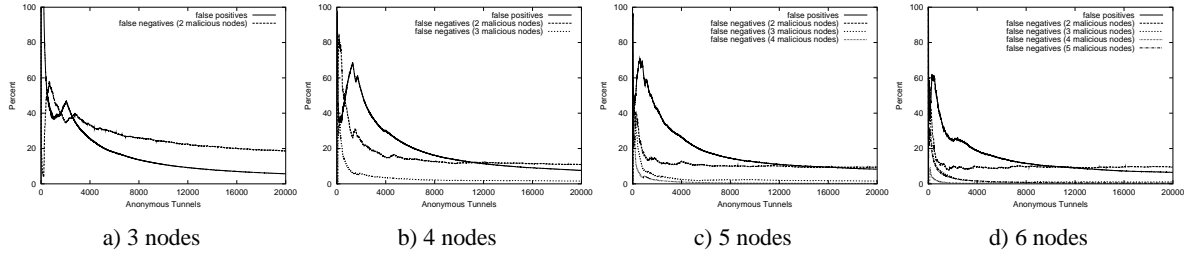
There is another interesting detail visible in figure 18. Comparing the correlation distributions in figures 18 b) and d), one can see a small peak of correlations stemming from the selections of malicious nodes in the range 0.002–0.006. These are exactly the correlations resulting from selections when the witness and the node delivering the selection collude. The correlation of the corresponding extended selection is quite low since the selection contains mainly well-behaving nodes. It is quite obvious that the initiating has no way to detect this malicious extended selection.

We conclude that it gets more difficult to detect malicious tunnels when taking malicious witnesses into account. On the other hand, the percentages of false negatives are still quite low, especially when there are many malicious nodes in an anonymous tunnel. A well-organized adversary would certainly make use of this attack and deliver fake selections whenever the node setting up the next hop and the corresponding witness are in the colluding set it controls. We will therefore include the malicious witnesses in all our further performance measurements.

#### 4.7 The Influence of the Tunnel Length

Looking at algorithm 2, it is reasonable to assume that the more extended selections of an anonymous tunnel we can test, the greater the probability to detect a bad tunnel. The reason is that we test each extended selection in step 5 of algorithm 2 but one bad extended selection is enough to reject the tunnel.

We now analyze how well our algorithm performs depending on the number of hops in a tunnel. Figure 19 shows the false positives and negatives when using 3–6 nodes in the anonymous tunnels. There are 100'000 nodes in the system and 10'000 of them are malicious and in the same colluding set. 20'000 anonymous tunnels are set up and we include malicious witnesses. The other parameters follow those of the previous examples.



**Figure 19:** False positives and negatives depending on the tunnel length

Figure 19 shows what we have expected: an anonymous tunnel with many hops increases the likelihood of detecting malicious nodes in the tunnel.

Table 9 shows the precise numbers of the false positives and negatives for figure 19 after having set up 20'000 anonymous tunnels.

**Table 9:** False positives and negatives (Figure 19)

	3 nodes	4 nodes
false positives	1'018 of 18'002 → 5.65%	1'218 of 16'099 → 7.57%
false negatives (2 malicious nodes)	<b>372 of 1'998 → 18.62%</b>	202 of 1'842 → 10.97%
false negatives (3 malicious nodes)		<b>32 of 2'059 → 1.55%</b>
	5 nodes	6 nodes
false positives	1'209 of 14'586 → 8.29%	845 of 13'040 → 6.48%
false negatives (2 malicious nodes)	157 of 1'649 → 9.52%	142 of 1'498 → 9.48%
false negatives (3 malicious nodes)	29 of 1'841 → 1.58%	18 of 1'646 → 1.09%
false negatives (4 malicious nodes)	<b>5 of 1'924 → 0.26%</b>	5 of 1'805 → 0.28%
false negatives (5 malicious nodes)		<b>8 of 2'011 → 0.40%</b>

What we should try to minimize at all costs are false negatives where all nodes except the initiator are malicious, i.e. anonymous tunnels of length 3 with 2 malicious nodes, those of length 4 containing 3 malicious nodes, those of length 5 with 4 malicious nodes, and those of length 6 where 5 nodes are in the colluding set. The corresponding figures are printed in boldface in table 9. Using only 3 nodes in a tunnel is a bad choice, as there is only one extended selection to examine during the setup of an anonymous tunnel, i.e. there is only one round of algorithm 2 for each anonymous tunnel. If that single extended selection passes the test during step 5 of algorithm 2, the anonymous tunnel is considered good. So if we falsely accept the selection from a malicious node, we have already lost. The 19% false negatives in table 9 have two reasons: (1) as there are 10% malicious nodes in the system, we will already miss 10% of all malicious tunnels due to colluding witnesses and corresponding nodes responsible for setting up the next hop (see section 4.6, and (2) since the initiator gets only one extended selection per tunnel, it takes relatively long until it has collected enough extended selections to make meaningful statements about the goodness or badness of an anonymous tunnel.

Using 4 or more nodes in the anonymous tunnel significantly improves our chances to detect a malicious tunnel of length  $n$  with  $n - 1$  malicious nodes. While the prime goal is to minimize the false positives when  $n - 1$  nodes in the anonymous tunnel are colluding, it is also desirable to recognize as many anonymous tunnels as possible where only the first 2 or 3 nodes are well-behaving. In general, it is easier to detect a tunnel consisting of only colluding nodes. The reason is the same as above: there are more malicious extended selections to test and consequently, the probability that at least one of the those not pass the test during step 5 of algorithm 2 is higher. On the other hand, using many nodes in anonymous tunnel also increases the time to set up an anonymous tunnel, the time to transport data from one end of the tunnel to the other, and the likelihood that a tunnel suddenly does not work anymore

because one of the nodes went down or observes other problems.

To summarize, one should not use only 3 nodes in her anonymous tunnels. Anything above 3 is reasonable, but the precise number of nodes to use is each user's own choice. If someone wishes to use 10 nodes in each tunnel to really keep the probability to eventually use a tunnel consisting of 9 malicious nodes extremely low, she is free to do so. However, 4 nodes in each tunnel gives already good protection from the adversary while 5 or 6 tunnels seem to be really enough to detect nearly all compromised tunnels. For the experiments to follow, we will stick to 5 nodes in an anonymous tunnel.

## 4.8 Assumptions of the Adversary

In our previous examples, we have always assumed that there is a single adversary that controls 10% of all nodes in the system. This view is of course too narrow and we therefore discuss in this section the capabilities of a reasonable adversary.

The adversary needs to control several nodes. The more nodes, the better his chances to control  $n - 1$  nodes in an anonymous tunnel consisting of  $n$  nodes without the initiator of the tunnel noticing this. One way is to operate all these nodes by himself, however, as we have mentioned in section 3.9.3, these nodes must at least reside in so many different subnets to offer valid selections during the setup of a nested encryption. Although this is beyond the scope of this technical report, we will extend the collusion detection mechanism in the sense that prefixes of IP addresses will be taken into account. This should effectively prevent an adversary from operating very many nodes in only a few subnets. So the nodes an adversary controls should reside in as many different subnets as possible. Simply owning 20 class C networks and operating 250 different nodes in each of them will not work. But having access to 5'000 computers in as many different subnets as possible is much harder than just controlling 20 class C networks. This could only be feasible for very large organizations operating worldwide in many different offices and having access to a variety of networks.

Another scenario could be that an organization wishing to control anonymous tunnels raises some money and starts offering 1'000 \$ per year to anybody that is willing to run a node at home, in his office, or anywhere where he has access to the public Internet. The nodes of those users that accept this offer would then form the colluding set of nodes. The organization would distribute a modified version of the software, which guarantees that only nodes from the coalition would be used in selections for next hop. Additionally, this software would carry out the attack described in section 4.6 if the corresponding witness also happens to be in the same coalition. The organization would also have to periodically distribute updated lists of the colluding nodes to make sure that each node has an up-to-date list of the other nodes in the coalition to make the whole attack as effective as possible. Finally, the software would make sure that each node delivers the information about the setup of anonymous tunnels it has collected to the organization and also all data sent forth and back when the node was the last hop in a tunnel. The organization would collect all this information and try to break the anonymity of the legitimate users of the system. Although this scenario would involve a large organizational overhead, we nevertheless believe it to be the most realistic scenario to collect many nodes from a variety of subnets in a coalition. For end users, the incentive to join such a coalition would be given by receiving money for running a piece of software on their computers and giving away some spare computing cycles and bandwidth, but no additional effort would be required. We could imagine that many users would make use of such an offer to earn some cheap money. On the other hand, finding many users that are willing to participate could most certainly not be done secretly and eventually, the Internet community would learn that there is indeed an organization trying to build a large coalition to break the anonymity of legitimate users of the system. Consequently, that organization would have to live with the fact that its efforts are publicly known and it is questionable what organization could accept this.

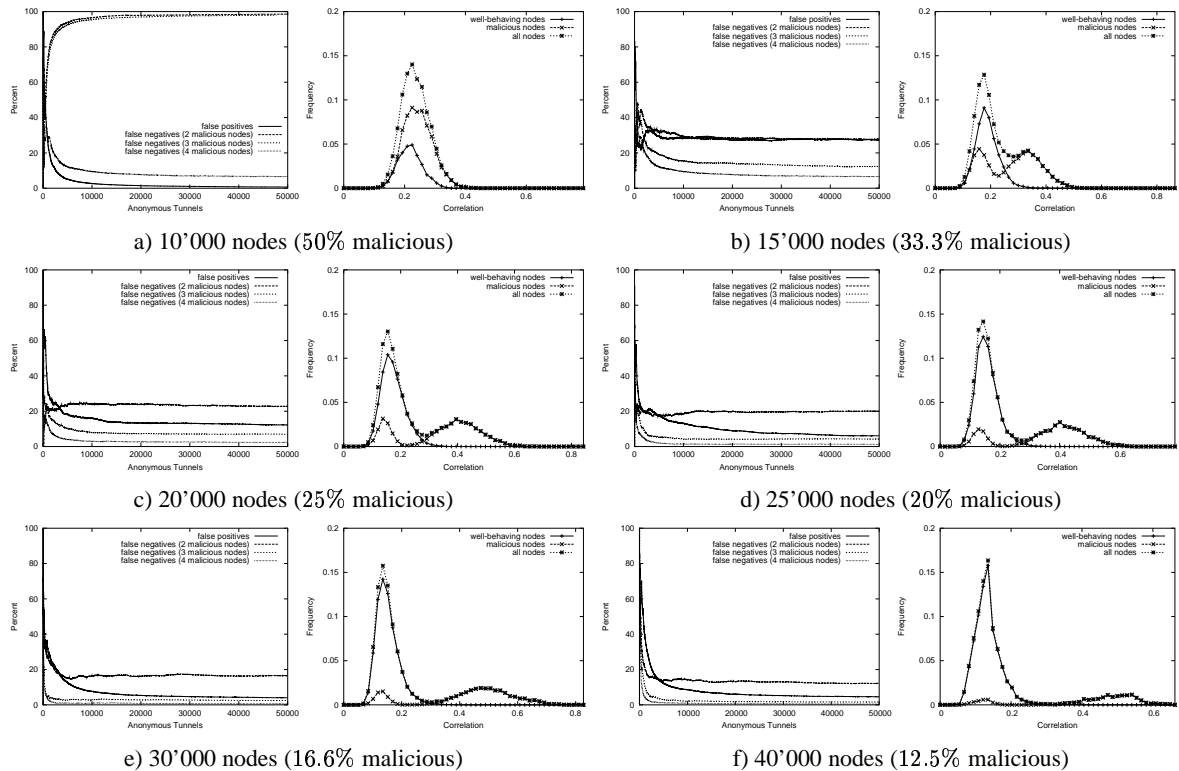
It is difficult to make an assumption for the number of nodes an adversary could control. But following our discussion above, we believe it is difficult to generate coalitions consisting of more than a few 1'000 nodes. Scenarios with 50'000 or even 100'000 as we examined in section 4.5 are not very real-

istic in our opinion. In our further performance measurements, we will therefore not consider coalition consisting of more than 10'000 nodes any more.

#### 4.9 Limits with a Single Adversary

In this section, we evaluate the limits of the basic model to detect malicious tunnels. We expect that the larger the relative number of node in the colluding set, the more difficult it gets to detect malicious tunnels. One reason is that with more malicious nodes, the relative number of tunnels that do not contain less than 2 malicious nodes becomes quite small. For instance, when using 5 nodes in a tunnel and half of the nodes are colluding, then the probability that any anonymous tunnel does not contain more than one malicious node is  $(0.5)^3 = 0.125$ , i.e. only 1 out of 8 tunnels is not malicious. Similarly, the probability that any selection the initiator receives is from a well-behaving node is only  $(0.5 + (0.5)^2 + (0.5)^3)/3 = 7/24 = 0.291\bar{6}$ , which means the internal table contains much more extended selections from malicious nodes than from well-behaving ones. Consequently, the shape of the correlation distribution is no longer dominated by a peak resulting from the extended selections of well-behaving nodes but by the correlations resulting from malicious extended selections. Another reason is that more malicious nodes means more malicious witnesses and consequently more selections that contain fake (i.e. well-behaving instead of malicious) nodes. As we have seen before, these extended selections containing fake nodes give correlations that are in the range of those resulting from non-malicious correlations. Consequently, the correlation distribution resulting from extended selections of malicious nodes “moves to the left”.

Figure 20 shows the false positives and negatives and the corresponding correlation distribution when the colluding set is fixed to 5'000 nodes and the total number of nodes varies from 10'000–40'000.



**Figure 20:** Performance depending on the relative number of colluding nodes

We can see immediately that our model does not seem to work when 50% of all nodes in the system are malicious (figure 20 a)). Looking at the corresponding correlation distribution, this is not surprising,

as the distributions of both well-behaving and malicious nodes are approximately in the same range (0.005–0.025) and the peak resulting from the malicious extended selections is much more dominant than the other. The sum of the two distributions results in one big peak that looks similar as if there were no malicious nodes at all in the system. Consequently, we have nearly 100% false negatives. The reasons why the two peaks are nearly in the same range are those discussed above: (1) most selections are offered by malicious nodes, which results in the dominant peak and (2) half of these malicious selections contain fake nodes, which causes the peak from the malicious extended selection to be more to the left than it would be if the witnesses were never malicious.

The smaller the number of nodes in the colluding set, the better our model works and the better out changes to detect malicious tunnels. The peak produced by the well-behaving nodes becomes more and more dominant and the correlations from malicious nodes move more and more to the right in the distribution. In addition, the probability that the witness and the corresponding node setting up the next hop are in the same colluding set decreases as the size of the colluding set gets smaller. This reduces the percentage of fake extended selections the initiating node receives, which is clearly visible in figure 20 as the small left peak of the malicious correlations gets smaller and smaller from b) to f). This improves the initiating node's changes to detect a malicious tunnel.

As already discussed in section 4.6, we cannot expect to detect a malicious tunnel if there is a malicious witness whenever a malicious node is setting up the corresponding next hop. With  $t$  nodes in the system and  $m$  of them build the colluding set, the probability that there is a malicious witness during the setup of a hop is  $\frac{m}{t}$ . If the tunnel consists of  $n$  nodes in total and  $i$  of them are malicious, then the best we can expect for the percentage of the false negatives is

$$p = \left(\frac{m}{t}\right)^{i-1}, \text{ where } 2 \leq i < n \quad (6)$$

With one third of all nodes being in the same colluding set, the best we can expect for the false negatives when there are 4 malicious nodes in a tunnel is  $p = (0.\bar{3})^3 = 0.037$ . According to figure 20, we get to about 6% false negatives after 100'000 tunnels have been set up, which is already relatively close to the optimum of 3.7%. The reason why we are still a little off the optimum is not surprising when looking at the correlation distribution in figure 20 b) and noticing that the overlapping range of the well-behaving and malicious nodes is significant. If the percentage of colluding nodes gets smaller, this overlapping range gets smaller, too and as a result, the false negatives gets closer to its maximum according to formula 6: with 25% malicious nodes, we get to about 2% and the optimum is  $p = (0.25)^3 = 1.56\%$ , and with 12.5% malicious nodes, we get to about 0.23% with an optimum of  $p = (0.125)^3 = 0.195\%$ .

To conclude, we state that the smaller the relative number of colluding nodes, the better the model of detecting malicious tunnels works. If more than about a third of all nodes in the system are in the same colluding set, then the model fails as the correlations stemming from well-behaving and malicious nodes are no longer distinguishable in the correlation distribution.

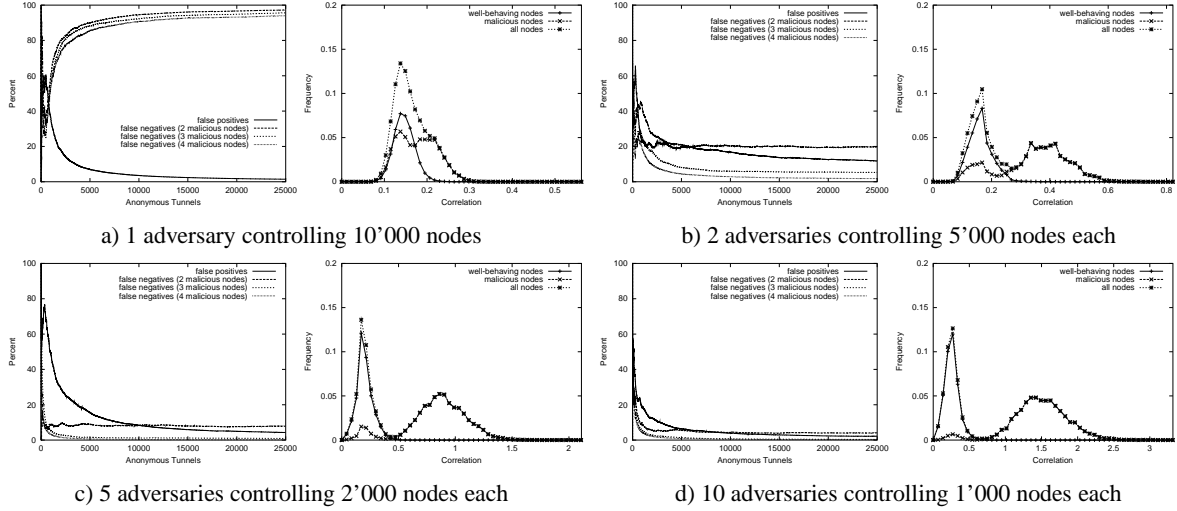
#### 4.10 Multiple Adversaries

Until now, we have always assumed that there is a single adversary controlling a single set of colluding nodes. However, in a real-world scenario, it is also possible that there are multiple, independently operating adversaries.

Facing two independent adversaries, each of them controlling  $m/2$  nodes should give us better changes to detect malicious tunnels than against one adversary controlling all  $m$  nodes. The reason is that both smaller sets can only choose among half as many nodes they offer in their selections when setting up the next hop, which requires fewer anonymous tunnels to set up for the initiator until the correlations from malicious extended selection become visible in the correlation distribution. A second reason

is that the probability that the node setting up the next hop and the corresponding witness is halved, which results in fewer extended selections that contain fake nodes.

Figure 21 illustrates the performance one can expect when there are 25'000 nodes in the system and 10'000 (40%) of them are malicious. We divide the malicious nodes among 1, 2, 5, and 10 different adversaries, whereas each adversary controls the same number of nodes.



**Figure 21:** Performance with 40% malicious nodes divided among the adversaries

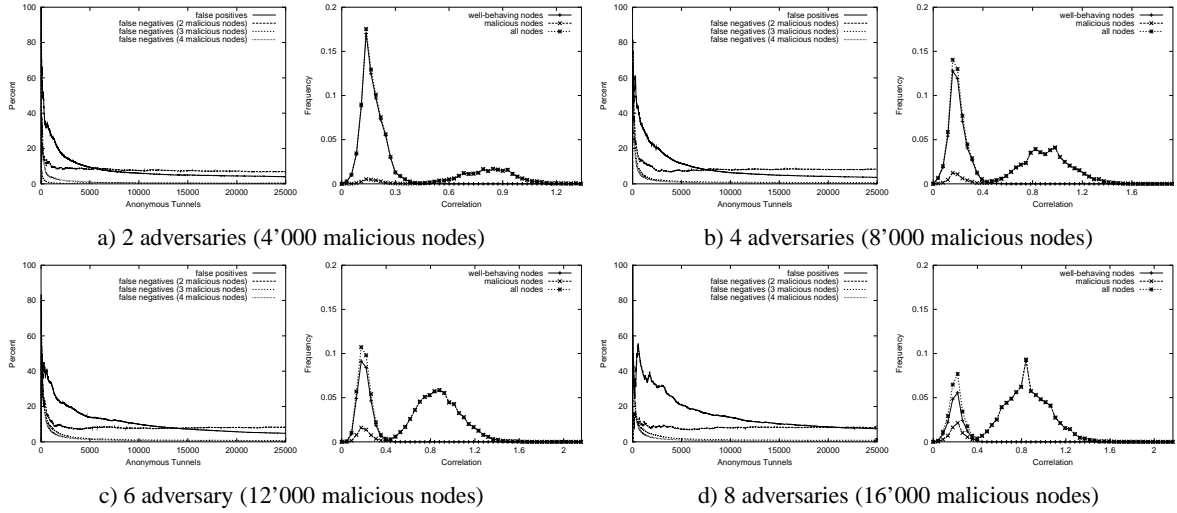
With one adversary controlling all malicious nodes (figure 21 a), our system fails again, which is not surprising following our discussion from section 4.9. However, if the 10'000 malicious nodes are divided among multiple adversaries, our changes to detect malicious tunnels get much better as figure 21 b–d illustrates. Even if the 10'000 nodes are split among two adversaries, only very few tunnels consisting of 4 malicious nodes remain undetected. Here again the main reason why the detection of malicious tunnels works better the more adversaries there are is directly visible from the correlation distributions, where the peaks resulting from the extended selections of well-behaving and malicious nodes get separated more and more as the number of adversaries increases.

Another question to answer is what happens if the number of nodes an adversary controls remains fixed but the number of adversaries is varied. Figure 22 illustrates the performance one can expect when there are 25'000 nodes in the system, an adversary controls always 2'000 nodes, and the number of adversaries goes from 2–8.

We can see that the model works very well when there are many adversaries and each of them controls 8% of the 25'000 nodes. Looking at figure 22 one can see that although 16'000 nodes are malicious (64%), nearly all of them can be recognized while keeping the number of false positives low. The correlation distribution illustrates nicely why the model works so well: Even with many malicious nodes, the peaks stemming from extended selections of well-behaving and malicious nodes remain clearly separated. As long as the left peak does not diminish too much, the model should always work well.

#### 4.11 Dependence on the Popularity of the Nodes

There is one final variation we have not looked at yet, and that is the influence of the popularity of the nodes. In all our experiments, we have always used that this popularity follows a negative exponential distribution such that the most popular nodes are about 50 times as popular as the least popular nodes. We have never given strong arguments that such an assumption has any validity, but it is probably safe to assume that not all nodes would be equally popular in a real life environment due to their connectivity, computing cycles they have to spare, reliability and so on. Nevertheless, as we cannot make predictions



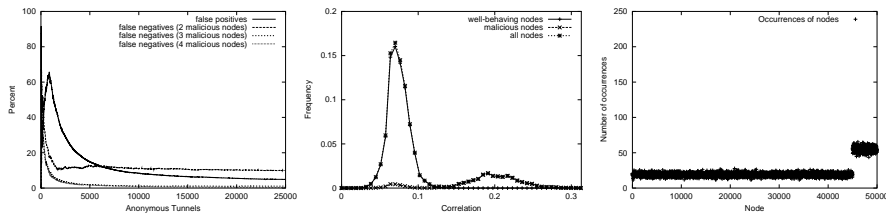
**Figure 22:** Performance depending on the number of adversaries

about the popularities of the nodes in our system, we will demonstrate how the model to detect malicious tunnels works depending on the popularities of the nodes.

We will always use a system with 50'000 nodes where there is a single adversary controlling 5'000 nodes, and set up 25'000 anonymous tunnels. We always depict the false positives and negatives, the correlation distribution, and how many times each node has been selected. The well-behaving nodes are always identified with numbers 0–44'999 and the malicious ones with 45'000–49'999.

#### 4.11.1 Uniform Distribution

If every node in the system is equally popular, it takes relatively long until the distribution correlation gets its typical shape. This follows directly from the discussion in section 4.3 where we stated that correlations  $c > 0$  for new extended selections only show up if the internal table contains at least one extended selection that has a node in common with the new extended selection. Figure 23 depicts the results when each node is equally popular.



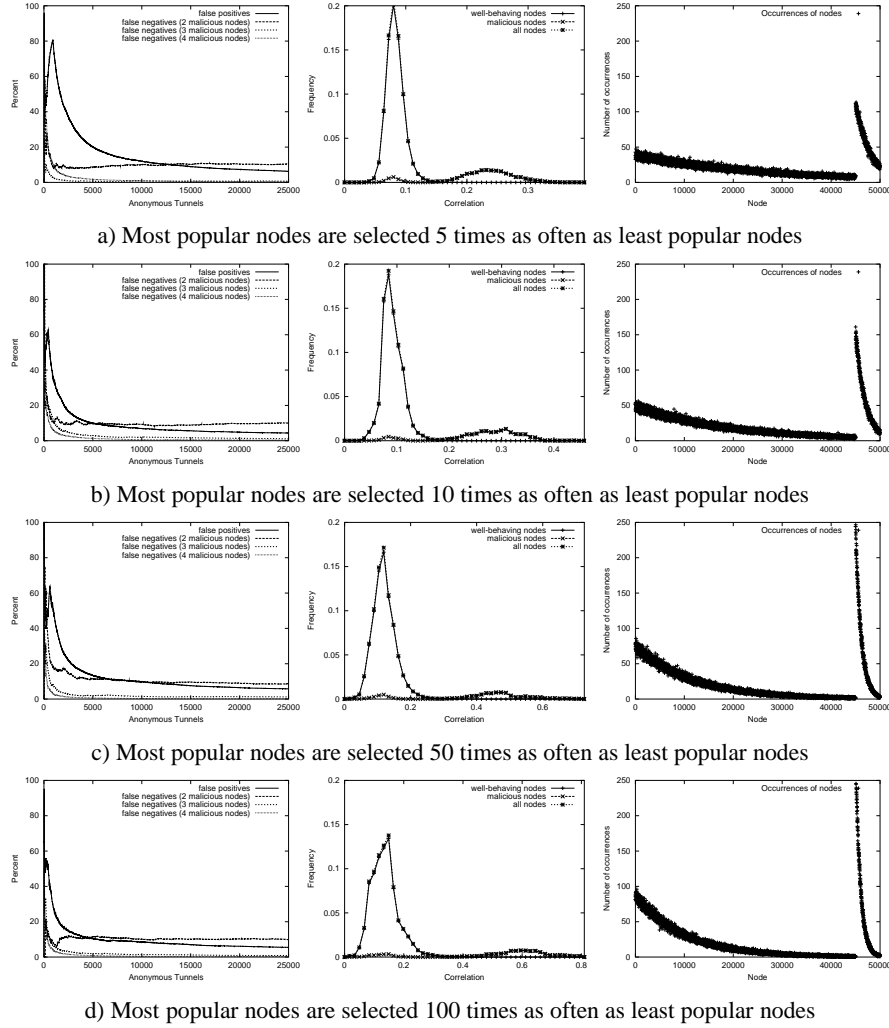
**Figure 23:** Performance with a uniform distribution of the popularities of the nodes

The results are similar as those we have seen in previous examples. Note that the malicious nodes (45'000–49'999) occur more frequently than the well-behaving ones. The reason is that whenever a malicious node is offering the selection, it contains only other malicious nodes (except when the witness is malicious, too), whereas a well-behaving node is selecting each of the 49'999 other nodes with the same probability.

#### 4.11.2 Negative Exponential Distribution

Although we have performed several experiments with this distribution before, the parameters were always chosen such that the most popular nodes are picked 50 times as often as the least popular nodes.

Here we will analyze how our model works when the parameters of the negative exponential distribution are varied. We examine the performance when the most popular nodes are selected 5, 10, 50, and 100 times as often as the least popular ones. Figure 24 illustrates the results.



**Figure 24:** Performance with a negative exponential distribution of the popularities of the nodes

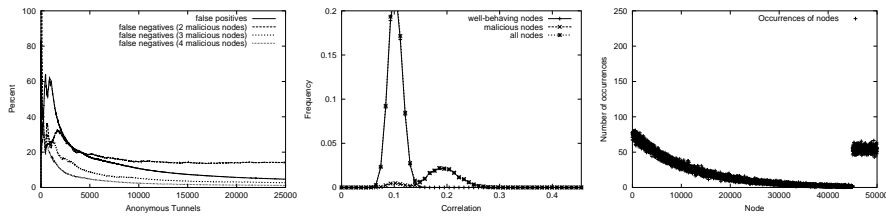
Here again, the model works very well independently of the parameters of the negative exponential distribution. Looking at the false positives and negatives in figure 24, then we can see that the lines approach 0 faster if the difference in the popularities of the most and least popular nodes is bigger. The explanation is the same as in section 4.11.1: the more uniformly the popularities of the nodes is, the longer it takes until a new extended selection has nodes in common with the extended selections in the internal table. This is also visible in the correlation distributions of figures 23 and 24: if the popularities of the nodes are more uniformly distributed, the peaks resulting from extended selections of well-behaving and malicious nodes are together more closely.

Looking at the number of occurrences of each node, one can see that both well-behaving and malicious nodes follow a negative exponential distribution. For the same reason as above, the malicious nodes are generally selected more frequently because of their preference in selections of malicious nodes. Nevertheless, the most popular well-behaving nodes are picked much more frequently than the least popular malicious nodes. This is why we cannot simply use the number of occurrences of each node to determine whether it is malicious or not.

#### 4.11.3 Negative Exponential Distribution with a more clever Adversary

When examining the influence of a negative exponential distribution we have assumed that the malicious nodes' popularities follow the same distribution as the well-behaving nodes. On the other hand, we have just seen that if the nodes are all equally popular then several anonymous tunnels are required until the correlations of extended selections produce values  $c > 0$ . Therefore, a clever adversary would try to make sure that his malicious nodes are all equally popular. As a consequence, it should take longer for the initiating node to reliably detect malicious tunnels.

Figure 25 illustrates the performance when the distribution of the popularities of the well-behaving nodes follows again a negative exponential distribution such that the most popular nodes are picked 50 times as often as the least popular nodes. The popularities of the malicious nodes, however, are uniformly distributed. There are again 50'000 nodes, 5'000 of them are malicious, and the probability that a well-behaving node picks a malicious node to be used in a selection or as a witness is 10%.



**Figure 25:** Performance with a more clever adversary

Comparing figure 25 with figure 24, we can immediately see that our assumption that it takes longer until the initiating node is able to make precise statements about anonymous tunnels was correct. But figure 25 reveals more: looking at the correlation distribution tells us that the correlations from extended selections of well-behaving and malicious nodes are partly overlapping, and consequently, the false positives and negatives are generally a little worse than they were in figures 23 and 24. Comparing this with figure 20, one could think that there are more than 10% malicious nodes involved. The explanation is that since the popularities of the well-behaving nodes follow a negative exponential distribution, the 50% most popular nodes are responsible for about 85% of all occurrences of well-behaving nodes as hops, witnesses, or as part of selections of well-behaving nodes. Each of the malicious nodes, however, occurs with the same probability as every other malicious node. According to formula 1, the size of the internal table depends on the number of different nodes the initiator has seen. After a while, the initiator has seen nearly all of the 50'000 nodes and – using a selection size of 14 (see table 4) – will use an internal table containing about  $n = 2 \cdot 50'000 \cdot (15/14) = 107142$ . However, due to the fact that some unpopular well-behaving only show up rarely, the table contains nearly all of the 5'000 malicious nodes but less than 30'000 of the 45'000 well-behaving ones. As result, there seem to be more than 10% malicious nodes at the initiator's side, and this effect is visible in the correlation distribution.

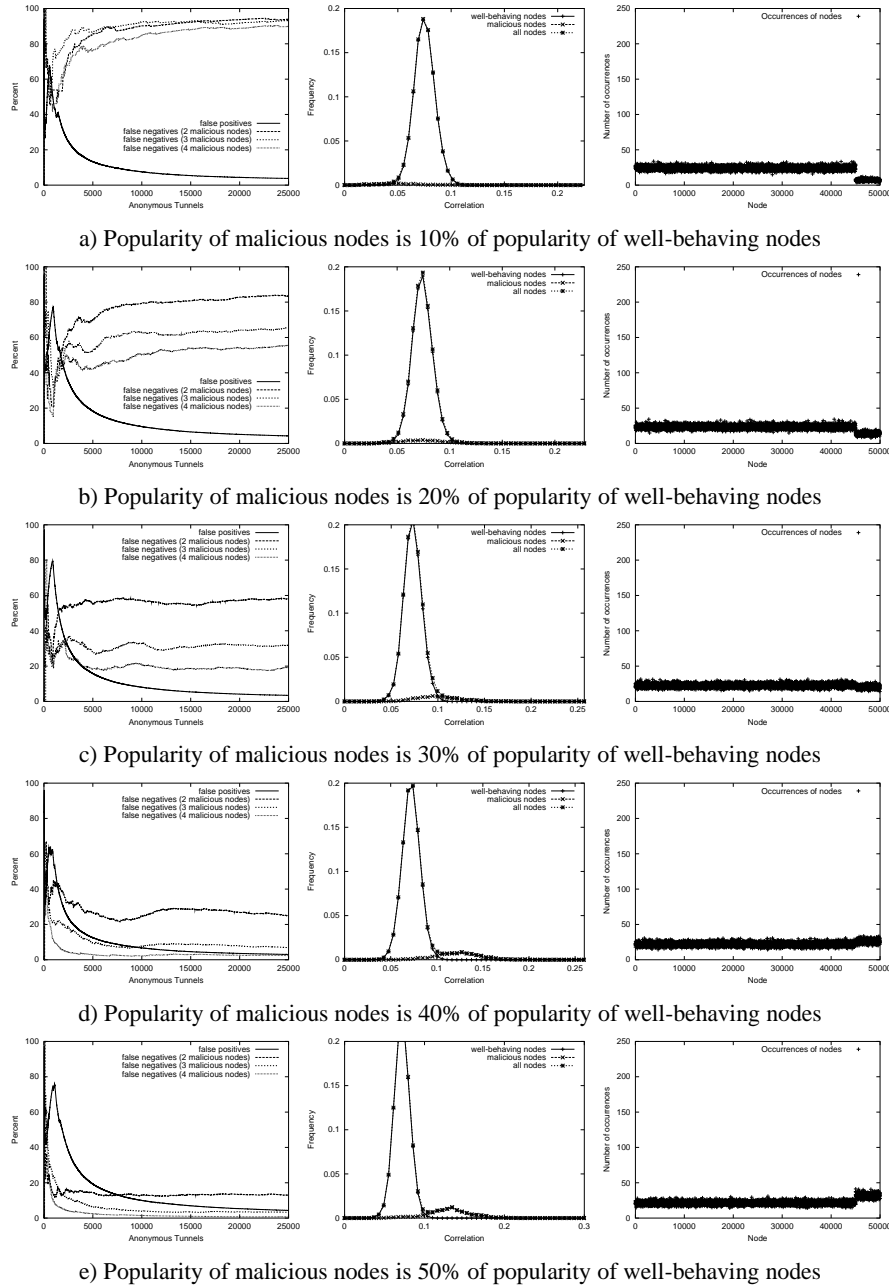
Of course we could increase the size of the internal table, but this would also increase the overhead to compute the correlation of each new selection. In addition, making the table bigger also implies that extended selections remain in the table for a long time, which gives a less accurate picture of the current situation of the system. Further more, looking at figure 25 shows that the model still works very well even against this more clever adversary and the increased complexity imposed by a bigger internal table would simply not be worth the minor improvement we would get. We therefore continue using formula 1 to determine the size of the internal table.

We briefly summarize the outcome of the experiments performed in this section, as they have delivered some interesting and important results. Whenever the popularities of the malicious nodes are more uniformly distributed than those of the well-behaving ones, then at the initiator's side this looks as if there is a bigger percentage of malicious nodes in the system than there actually is. The reason is that the internal table contains most of the malicious nodes but several unpopular well-behaving are missing.

#### 4.11.4 Uniform and Negative Exponential Distribution with a silent Adversary

Up to now, we have assumed that if the adversary controls about 10% of all nodes, then the probability that his nodes are selected as hops, witnesses, or as part of selections by other hops is also about 10%. We have seen that this makes it very difficult for the adversary to control all  $n - 1$  hops of a tunnel of length  $n$  without being detected by the initiator. The reason is that there are many extended selections consisting of only malicious nodes in the internal table, and this results in relatively big values for the correlation according to algorithm 1. If the adversary manages to keep the number of extended selections in the initiator's node small, then the chances his malicious tunnels remain undetected should improve.

Figure 26 illustrates the performance when the popularities of all well-behaving nodes is the same and those of the malicious nodes is smaller. There are again 50'000 nodes, 5'000 of them are malicious.



**Figure 26:** Performance with a uniform distribution of the popularities of the nodes and a silent adversary

Figure 26 confirms what we have already suspected: if the adversary manages to keep the popularities of his nodes small, then many malicious tunnels remain undetected. Especially when the popularity of malicious nodes is only 10% of the popularity of the well-behaving nodes (figure 26 a), it is virtually impossible to detect malicious tunnels as the correlation distributions of well-behaving and malicious nodes overlap significantly. If the malicious nodes' increases to 20%, there still remain about 60% of the anonymous tunnels containing 4 malicious nodes undetected. While this seems to be a lot, one has to bear in mind that it is not so much the percentage of false negatives that is important, but the percentage of completely malicious tunnels of all anonymous tunnels that remain undetected by the initiator that should be minimized. Table 10 lists the total percentage of completely malicious tunnels corresponding to figure 26 when the initiator sets up 100'000 anonymous tunnels.

**Table 10: Percentage of completely malicious tunnels (figure 26)**

relative popularity of malicious nodes malicious	number of completely malicious tunnels set up	number of completely malicious tunnels undetected	false negatives	percentage of completely malicious tunnels undetected
10%	1'141	1'036	96.54%	<b>1.04%</b>
20%	2'203	1'236	56.11%	<b>1.24%</b>
30%	3'163	607	19.19%	<b>0.61%</b>
40%	4'195	77	1.84%	<b>0.08%</b>
50%	5'240	17	0.32%	<b>0.02%</b>
100% (figure 23)	10'119	16	0.16%	<b>0.02%</b>

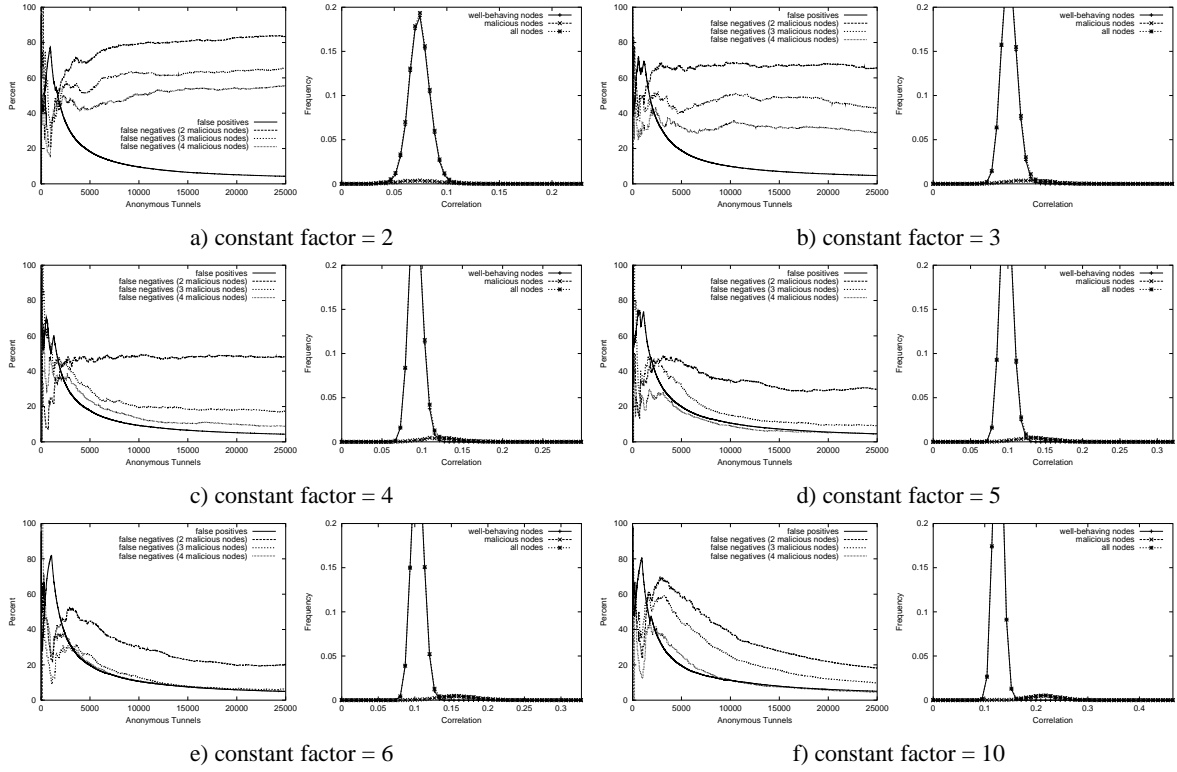
The most interesting column of table 10 is the last one. It tells what percentage of all anonymous tunnels the initiator sets up the adversary is likely to compromise. It seems that the adversary's chances are best if he makes sure that his nodes are about 5 times less popular than the well-behaving ones. Although he then manages only to completely compromise about 2% of all anonymous tunnels, about 60% of them will be accepted by the initiator as non-malicious. Consequently, the adversary completely controls about 1.24% of all anonymous tunnels the initiator uses to send and receive data anonymously. If the popularity of the malicious nodes is only 10% of that of the well-behaving ones, then the initiator will accept nearly all of the tunnels controlled by the adversary, but nevertheless this strategy would give the adversary control over only about 1% of the initiators anonymous tunnels. If the adversary's strategy is more aggressive by making sure the popularity of the malicious nodes is higher, his success rate also decreases as the initiator is now able to detect most malicious tunnels.

The reason why so many compromised tunnels remain undetected if the popularities of the malicious nodes is low is that there are only a few malicious extended selections in the internal table. It should therefore be possible to increase the initiator's chances to detect malicious tunnels if the size of the internal table is increased. Figure 27 illustrates this when the constant factor in formula 1 is increased from 2 to 10. We perform the measurement from figure 26 b) with the popularities of malicious nodes at 20% of the popularities of the well-behaving nodes.

Increasing the size of the internal table does indeed help. The bigger the internal table, the more clearly separated the correlation distributions of well-behaving and malicious nodes are. Table 11 lists again the total percentage of fully compromised tunnels that remain undetected by the initiator after having set up 100'000 tunnels.

Doubling the size of the internal table, i.e. using a constant factor of 4 instead of 2 in formula 1, already brings down the total percentage of completely compromised tunnels by a factor of about 8.

In section 4.11.3, we have learned that if the popularities of the malicious nodes are more uniformly distributed than those of the well-behaving ones, then the adversary's chances that malicious tunnels remain undetected by the initiator are higher. In this section, we have seen that if the popularities of malicious nodes are smaller than those of the well-behaving ones, then his chances increase, too. By



**Figure 27:** Performance with a uniform distribution of the popularities of the nodes and a silent adversary when varying the constant factor of formula 1

**Table 11:** Percentage of completely malicious tunnels (figure 27)

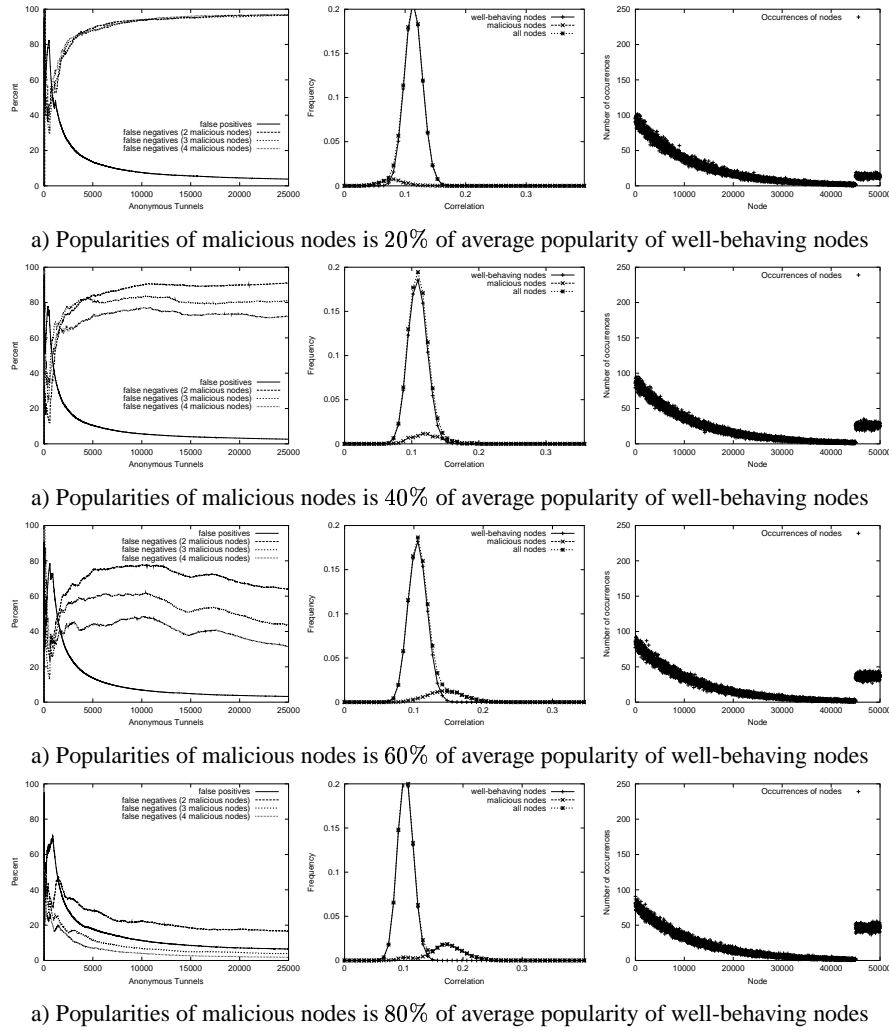
constant factor in formula 1	number of completely malicious tunnels set up	number of completely malicious tunnels undetected	false negatives	percentage of completely malicious tunnels undetected
2	2203	1236	56.11%	<b>1.24%</b>
3	2180	479	21.97%	<b>0.50%</b>
4	2144	160	7.46%	<b>0.16%</b>
5	2186	68	3.11%	<b>0.07%</b>
6	2131	36	1.69%	<b>0.04%</b>
10	2075	23	1.11%	<b>0.02%</b>

combining these two results, we expect that the total percentage of malicious tunnels should and therefore the adversary's chances to compromise anonymous tunnels should be bigger than in figure 26.

Figure 28 depicts the performance when the distribution of the popularities of the well-behaving nodes follows a negative exponential distribution such that the most popular nodes are picked 50 times as often as the least popular nodes. The popularities of the malicious nodes are uniformly distributed. There are again 50'000 nodes, 5'000 of them are malicious, and the popularities of the well-behaving nodes range from 20%–80% of those of the average well-behaving ones.

Table 12 shows again at the total percentage of fully compromised tunnels that remain undetected by the initiator after having set up 100'000 tunnels.

In this scenario, the adversary manages to control about 3% of all tunnels the initiator sets up when keeping the popularities of the malicious nodes in the range of 40%–60% of the average popularity of the well-behaving ones. Let's analyze again what the initiator gains by increasing the size of the



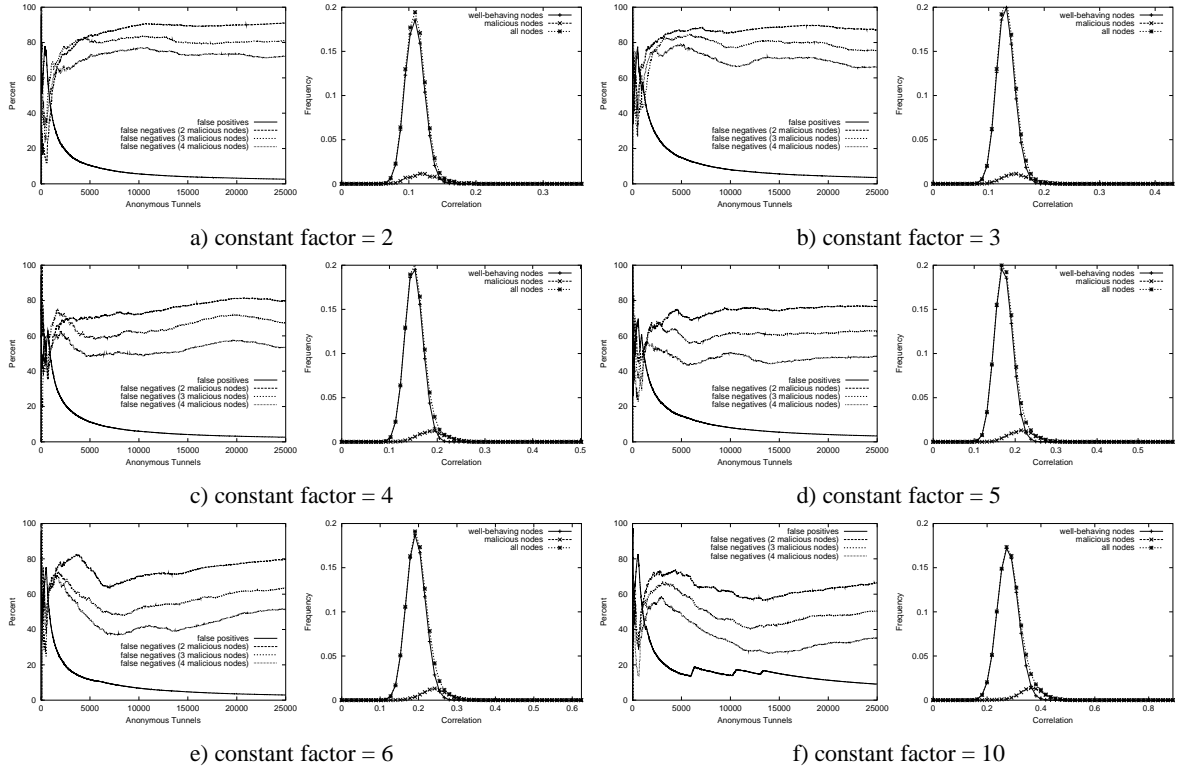
**Figure 28:** Performance with a negative exponential distribution of the popularities of the nodes and a clever and silent adversary

**Table 12:** Percentage of completely malicious tunnels (figure 28)

relative popularity of malicious nodes malicious	number of com- pletely malicious tunnels set up	number of com- pletely malicious tunnels undetected	false negatives	percentage of com- pletely malicious tunnels undetected
20%	2'101	2'074	98.71%	<b>2.07%</b>
40%	4'300	3'005	69.88%	<b>3.01%</b>
60%	6'178	1'083	17.53%	<b>1.08%</b>
80%	8'175	70	0.86%	<b>0.07%</b>
100% (figure 25)	10'030	40	0.40%	<b>0.04%</b>

internal table. Figure 29 illustrates this when the constant factor in formula 1 is increased from 2 to 10. We perform the measurement from figure 28 b) with the popularities of malicious nodes at 40% of the popularities of the well-behaving nodes.

To compare how much we could increase the initiator's chances to detect malicious tunnels, table 13 lists the total percentage of fully compromised tunnels that remain undetected after having set up 100'000 tunnels.



**Figure 29:** Performance with a negative exponential distribution of the popularities of the nodes and a clever and silent adversary when varying the constant factor of formula 1

**Table 13:** Percentage of completely malicious tunnels (figure 27)

constant factor in formula 1	number of completely malicious tunnels set up	number of completely malicious tunnels undetected	false negatives	percentage of completely malicious tunnels undetected
2	4'300	3'005	69.88%	<b>3.01%</b>
3	4'184	2'771	66.23%	<b>2.77%</b>
4	4'340	2'475	57.03%	<b>2.48%</b>
5	4'218	2'521	59.77%	<b>2.52%</b>
6	4'199	2'542	60.54%	<b>2.54%</b>
10	4'311	2'529	58.66%	<b>2.53%</b>

It seems we have reached the limits of our model. Even increasing the size of the internal table does not significantly reduce the total percentage of fully compromised tunnels and allows the adversary to control about 2.5% of all anonymous tunnels the initiator sets up. While this seems to be a lot, one should bear in mind that this is only possible if the popularities of the well-behaving nodes are distributed very non-uniformly and the popularities of the malicious nodes are relatively low and distributed as uniformly as possible. While maintaining the malicious nodes popularities low should not be too difficult, it will not be easy for an adversary to make sure that all malicious nodes are approximately equally popular and their popularities are adapted to the current state of the whole system to make the attack efficient.

To conclude, we remember that for the adversary, the best strategy is to make sure that all nodes he controls are about equally popular but less popular than the average well-behaving node. To counter this attack, the popularities of all well-behaving nodes should be as similar as possible. Although this will never be the case in a real-world system, each well-behaving node can contribute to this by changing its

neighbor nodes frequently and offering nodes from a vast variety in its selections.

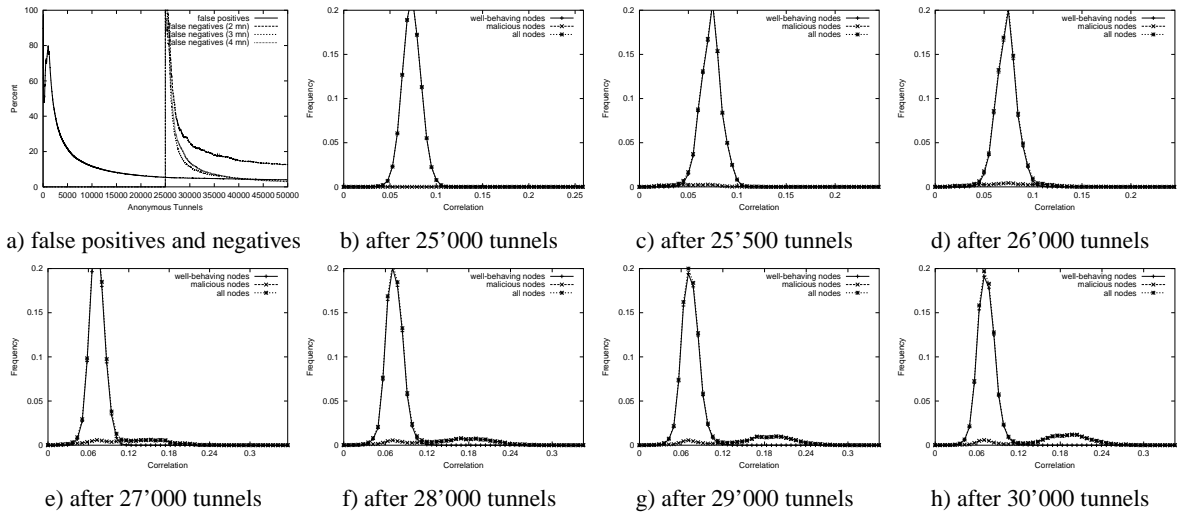
Using a bigger constant factor in formula 1 can help to reduce the adversary's changes to get full control over anonymous tunnels, but, as seen in figure 29, this has its limits, too. As for now, we still believe that a factor of 2 is a good compromise between efficiency and protection from the adversary.

## 4.12 Behavior in a Dynamic Environment and Scalability

All the performance measurements we have done so far were based on a system with  $n$  nodes where all of these  $n$  nodes are there from the beginning and do not leave the system. In reality, nodes come and go, some of them staying for a long time and some of them showing up only occasionally. New nodes will pop up every now and then and other nodes will leave and never come back. In this section, we will examine how this affects our basic model to detect malicious tunnels.

### 4.12.1 A Set of Colluding Nodes Suddenly Appears

In a first experiment, we analyze what happens if a set of colluding nodes suddenly appear and start trying to compromise anonymous tunnels. We use a system consisting of 50'000 nodes, 5'000 of which are malicious and colluding. We set up 50'000 anonymous tunnels, but the malicious nodes will only appear after 25'000 tunnels have been set up. We assume that all nodes are equally popular. Figure 30 shows the false positives and negatives and the correlation distribution after having set up 25'000–30'000 nodes.



**Figure 30:** Performance when the colluding nodes appear after having set up 25'000 anonymous tunnels

Figure 30 a) shows that when the colluding nodes come into play at 25'000 tunnels, the false negatives go up to 100% before rapidly dropping again after about 25800 tunnels. Looking at the correlation distribution explains this behavior: After 25'000 tunnels (figure 30 b), there is one single peak as there have been only well-behaving nodes in the system so far. When the malicious selections start to arrive at the initiating node, the corresponding correlations will be low in the beginning (algorithm 1). This is clearly visible in figure 30 c) where a small peak stemming from the malicious selections is visible, but the peak is very small and lies on the left of the big peak. When more malicious extended selections are accumulated by the initiator, the resulting correlations will bigger and bigger and consequently, the peak moves to the right and becomes more visible (figures 30 d–h).

Table 14 shows the precise numbers of the false positives and negatives for figure 19 after having set up 20'000 anonymous tunnels.

**Table 14:** *False positives and negatives (Figure 30)*

false positives	1'741 of 43'291 → 4.02%
false negatives (2 malicious nodes)	246 of 1'948 → 12.63%
false negatives (3 malicious nodes)	73 of 2'275 → 3.21%
false negatives (4 malicious nodes)	72 of 2'486 → <b>2.90%</b>

We can see that the adversary manages to fully compromise 72 anonymous tunnels before the initiator has collected enough information to recognize further compromised tunnels. Considering that the adversary suddenly appears with many malicious nodes (5'000), we think that 72 compromised tunnels is not much and conclude that our model copes very well with this situation.

#### 4.12.2 Scalability

In this section, we analyze the effort needed by the initiator to process a new extended selection. If we look at the main steps that must be carried out when a new extended selection arrives, we get the following:

1. Compute the correlation of the new extended selection according to algorithm 1.
2. Determine if the new extended selection is good or bad by applying algorithm 2.
3. Check if the new selection contains new nodes, i.e. nodes that have not been seen previously. The reason for this is that the node must keep track of the number of nodes it has seen so far.
4. Modify the correlation distribution according to formula 2.
5. Append the new extended selection to the internal table and remove old selections if this is needed according to formula 1.

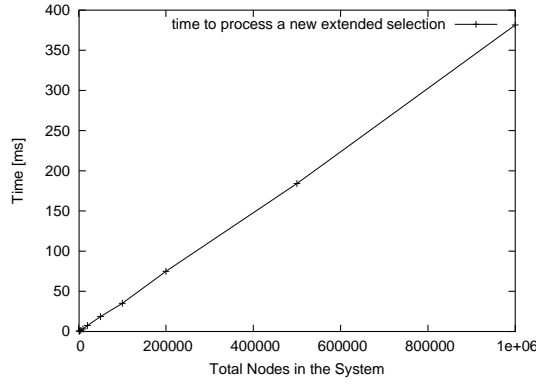
According to formula 1, the number of extended selections in the internal table is proportional to the number of different nodes  $n$  the initiating node has seen. Accordingly, the time it takes to perform step 1 is proportional to  $n$ , which means its complexity is  $o(n)$ . The list of previously seen nodes can be implemented as an ordered list, where searching and inserting can be done with complexity  $o(\log(n))$ . This implies that step 2 can also be performed with complexity  $o(\log(n))$ . The time needed to perform steps 2, 4, and 5 is independent of  $n$ . Step 1 dominates the whole time it takes to process a new extended selection and is proportional to the number of nodes the initiator has seen previously.

Figure 31 depicts how long it takes for a node to completely process an incoming extended selection. The tests are carried out on a system with a 1GHz AMD Athlon CPU, 256 MB RAM, running Linux as operating system with a 2.4.17 kernel. The software is written in Java and we use Sun's Java 2 SDK 1.4.

Figure 31 nicely exposes that the time it takes to process a newly arriving extended selection is indeed proportional to the number of nodes the initiator has seen. This is not a problem if the system consists of relatively few nodes, e.g. a couple of 10'000, and the corresponding time is only a few milliseconds. But as the number of nodes reaches several 100'000, we eventually run into problems, as the processing time increases to 100 milliseconds and beyond. The question is what can we do to keep the time to process an extended selection low after the initiator has seen possibly millions of nodes.

#### 4.12.3 Modeling a Dynamic Environment in more Detail

Before we try to solve our scalability problem, we define what we believe could be a realistic scenario in more detail. We have already given arguments why not all nodes will be equally popular, so we will continue modeling this with a negative exponential distribution where the most popular nodes are 50



**Figure 31:** Time to process a new extended selection

times as popular as the least popular ones. What we haven't taken into account so far is that nodes are not always available. Some will be up and running all the time, while others are only active occasionally. Furthermore, some users may just be curious and have a look at the system, start their node once, leave after half an hour, and never come back. In general, one can argue that the popularity of nodes is independent of their availability, but we argue that this is not the case. A node  $P$  gets popular by accepting being used by many other nodes. This is only possible if  $P$  has the necessary processing power and an adequate network connection, and this is independent of its availability. However, the fact that a node is present has also to be spread through the network, and as soon as  $P$  is no longer available, other nodes being connected to  $P$  realize this and no longer propagate the information about its presence. It is therefore more difficult for a node to get popular if it not often available. On the other hand, there are many unpopular nodes running on computers with poor network connections and therefore not able to handle many anonymous tunnels. While there may well be such unpopular nodes that are nearly always available, we can generally assume that many of them are only available when their operator is actually using the system to browse the web anonymously. We conclude that the availability of popular nodes is generally expected to be higher than that of unpopular nodes, but there are always nodes that do not follow this rule.

To model the availability of nodes, we assign an availability probability between 0 and 1 to each node. In a first step, we determine a parameter  $k$  which depends on the probability of the node. Assume there are  $n$  well-behaving nodes in the system and the nodes are sorted in descending order according to their popularity. The nodes are identified by an index  $i$  where  $0 \leq i < n$ . Node  $i$  is the most popular and node  $n - 1$  is the least popular node. For each node  $i$ , where  $0 \leq i < n$ , the parameter  $k$  is defined as

$$k(i) = 1 - e^{-4 \cdot (i+1) \cdot \frac{1}{n+1}} \quad (7)$$

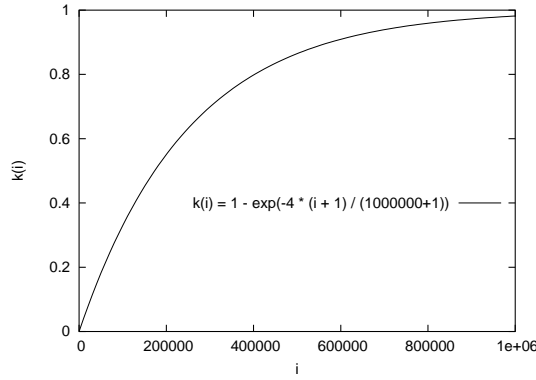
$k(i)$  is ascending from 0 to 1 where  $k(0)$  is a little bigger than 0 and  $k(n - 1)$  is a little smaller than 1. Figure 32 depicts the values of  $k(i)$  in a system with 1'000'000 well-behaving nodes.

To determine the availability of node  $i$ , we use another function with parameter  $k$ . The function is defined for  $0 \leq x \leq 1$  and is defined as

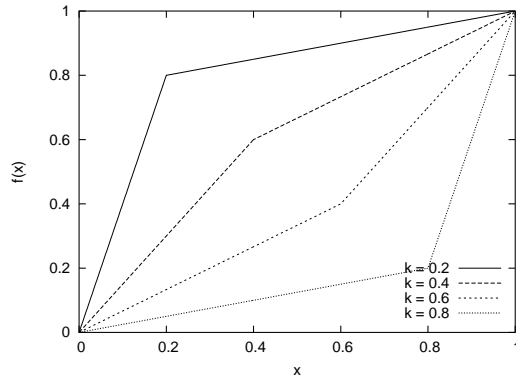
$$f(x) = \begin{cases} x^{\frac{1-k}{k}} & \text{if } x \leq k \\ \frac{x \cdot k + 1 - 2 \cdot k}{1-k} & \text{if } x > k \end{cases} \quad (8)$$

Figure 33 depicts the function  $f(x)$  for 4 different values of the parameter  $k$ .

Figure 33 shows that the bigger the parameter  $k$ , the bigger the values for  $f(x)$ . To determine the actual value of the availability  $a(i)$  of node  $i$ , a random number  $r(i)$ , where  $0 \leq r \leq 1$  is first determined



**Figure 32:**  $k(i)$  in a system with 1'000'000 well-behaving nodes



**Figure 33:**  $f(x)$  depending on the parameter  $k$

for each  $i$ . The resulting  $r(i)$  are distributed uniformly over the interval  $[0, 1]$ . Using  $i$ 's parameter  $k(i)$  (according to 7) together with  $r(i)$  in 8, we get

$$a(i) = \begin{cases} r(i) \frac{1-k(i)}{k(i)} & \text{if } r(i) \leq k(i) \\ \frac{r(i)k(i)+1-2k(i)}{1-k(i)} & \text{if } r(i) > k(i) \end{cases} \quad (9)$$

What have we achieved with  $a(i)$ ? According to figure 32, the more popular a node  $i$  is, the bigger  $k(i)$  gets. As a result, the more popular  $i$ , the bigger the values of the function  $f(x)$ , which we have seen in figure 33. If we now pick a random  $r(i)$  and use it in 9, we therefore guarantee that (1) more popular nodes will generally have an availability that is relatively high and vice versa, and (2) that it is still possible that some popular nodes will get low availabilities and vice versa. This is exactly what we wanted to achieve following our discussion above.

One could also model the same differently, and it is difficult to judge what model would be the best as long we so not exactly knows how a real system with real nodes behaves. For now, we believe that this model is good enough to perform further experiments. The reason for using the constant factor  $-4$  in the exponent of 7 is that this results in about 25% of all well-behaving nodes being active at any time.

The situation is different on the adversary's side. To be most effective, the adversary should try to keep the nodes he controls active all the time. This follows directly from our discussion in section 4.9. As it is very difficult to guarantee this, we assume that each of the adversary's nodes have an availability of 0.9. In addition, we have seen in section 4.11 that the attack is most effective if the malicious nodes are equally popular and if the popularities of the malicious nodes is generally kept low. We therefore

will assume a uniform distribution of the malicious nodes' popularities for our next experiments. The average popularity of a malicious node is 50% of the average popularity of the well-behaving nodes.

#### 4.12.4 Making the System Scalable

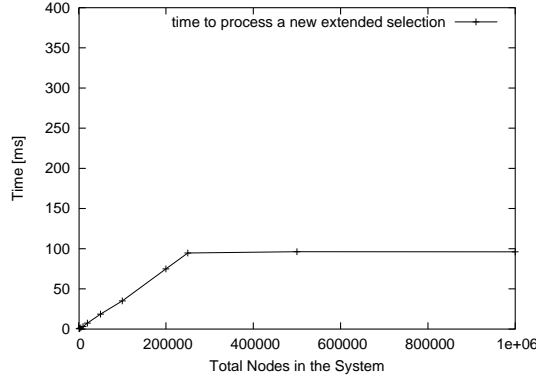
To achieve scalability in our system, we must not let grow the size of the internal table indefinitely, and this is exactly what we are doing. Following the discussion in the previous section, we state that some well-behaving nodes will always be selected more often as part of selections than others due to their availability and popularity. As a result, the extended selections in the internal table contain mainly well-behaving nodes with relatively high availability and popularity and malicious nodes at any time. Very many of the unpopular and less available nodes are not present at all in the internal table or only in one or two extended selections. This implies that the value of the correlation one computes according to algorithm 1 whenever a new extended selection is received is mainly determined by the more available and popular well-behaving and the malicious nodes, i.e. it seems as if the less popular nodes would not exist. But this also implies that we could probably ignore the nodes that show up only very occasionally and use a smaller value for  $n$  in formula 1 than the number of different nodes the initiator has already seen.

The idea is to base  $n$  not on all nodes the initiator has seen, but only on those it has seen frequently. The question is what nodes are to be considered frequently and thereby relevant for the value of  $n$ . One way is counting the number of times each node occurs as part of newly arriving selections. Since some nodes are selected much more frequently, it is probable that something like 15% of the nodes the initiator has seen so far are responsible for 80% of all occurrences. That could be the way to determine the value used for  $n$  in formula 1. So if the initiator had seen 1'000'000 nodes but 150'000 of these nodes were responsible for 80% of all occurrences of nodes in selections, then  $n = 150'000$  would be used in formula 1. However, the problem with this approach is that the initiator has to maintain information about all nodes it has seen so far. This needs more and more memory as the number of nodes grows and becomes a real problem when the initiator has seen very many nodes. An even bigger problem is maintaining the data structure to manage the number of times a node has already occurred. At the same time, we have to search for a node in that data structure, be able to dynamically insert new nodes, and to determine the most frequent nodes that are responsible for 80% of all occurrences. Not all of these requirements can be done efficiently at the same time, which results in a complexity of  $o(n_{total})$  to handle a selection, where  $n_{total}$  is the total number of nodes the initiator has seen. What we need is a way to roughly emulate this behavior without having to keep track of all nodes.

We have argued in section 4.8 that it is not likely that the number of nodes an adversary is able to control grows very much beyond 10'000 nodes. To reasonably resist such an adversary, the system would have to consist of approximately 50'000 well-behaving nodes (see section 4.9). Following our discussion above, it should therefore be enough to simply consider the 60'000 most frequently selected nodes, i.e. to use  $n = 60'000$  in formula 1 even if the number of different nodes the initiator has already seen is much bigger. We implement this as follows: there is an upper limit  $n_{limit}$  which is the biggest  $n$  the initiator wants to use in formula 1. As long as  $n \leq n_{limit}$ , nothing changes and the real number of different nodes the initiator has seen is used in formula 1. But as soon as  $n > n_{limit}$ ,  $n_{limit}$  is used in formula 1, which guarantees that the size of the internal table cannot grow beyond a certain limit.  $n_{limit}$  can be adapted by the user but for now, we will set it to 250'000. According to figure 31, this should keep the time to process a newly arriving selection below 100 ms on our test system.

What we haven't considered yet is what happens with nodes that once were part of the system but have left a while ago and possibly never come back. An initiator that has seen such a node would remember it forever and every node that is no longer present has an influence on  $n$ . As a result, the number of nodes seen can only grow, but never shrink. To cope with this, the initiator not only remembers the IP addresses of nodes it has seen, but also the time it has seen a node the last time. Whenever a node hasn't been seen for a while, e.g. a month, it is deleted from the list and  $n$  is reduced by one.

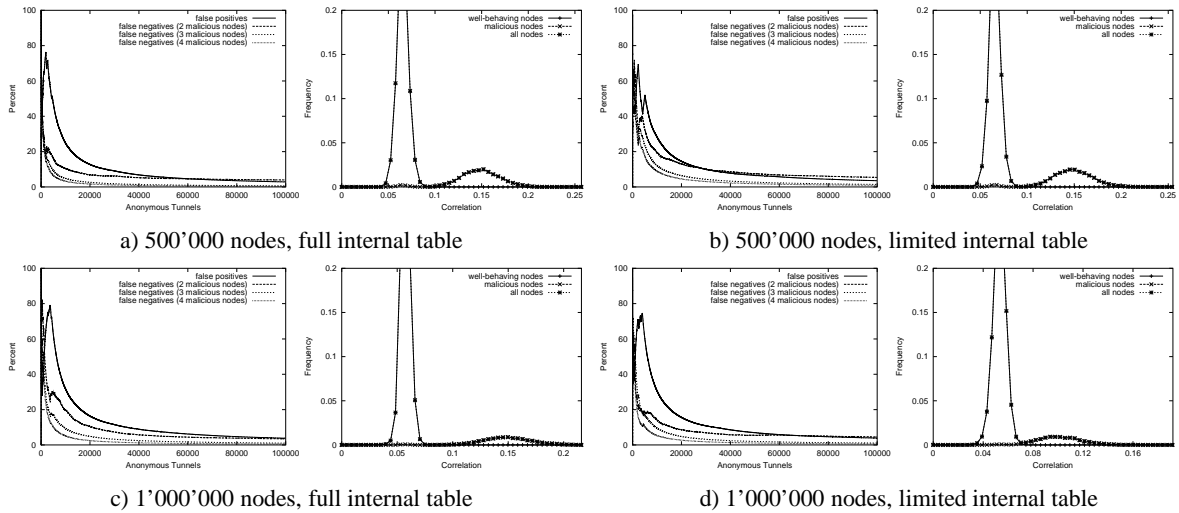
We first analyze the influence of the limited size of the internal table on the time to completely process an incoming extended selection. We use exactly the same setting as in figure 31. Figure 34 depicts the results.



**Figure 34:** Time to process a new extended selection with the limited internal table

One can see that the time still increases linearly as the number of nodes seen grows. After 250'000 nodes, however, the time remains more or less constant since the size of the internal table has reached its maximum. The limited size guarantees that the time to process a new extended selection stays below 100 ms.

We now want to check how much the limitation of the table size affects the performance of the whole system. To model the dynamic behavior of the system, we determine a new set of active nodes after every 100 anonymous tunnels the initiator has set up. For each node  $i$ , we pick a random number  $r(i)$  between 0 and 1. If the  $r(i) \leq a(i)$ , where  $a(i)$  is the availability of node  $i$  according to formula 9, then node  $i$  will be active during the setup of the next 100 anonymous tunnels, otherwise it will not be active. This implies that a node with a high availability will be active most of the time, while nodes with very small availabilities will only be active occasionally. Figure 35 depicts the performance for a system with 500'000 and 1'000'000 nodes. We assume three different adversaries where each of them controls 10'000 different nodes. All other parameters follow those specified in section 4.12.3.

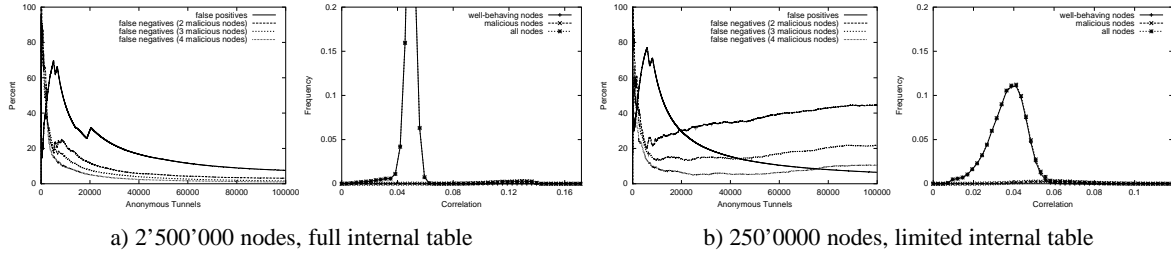


**Figure 35:** Performance with a limited internal table size

Figure 35 shows that limiting the size of the internal table does indeed work. The false positives and negatives are nearly identical independently whether the full or the limited internal table size is used.

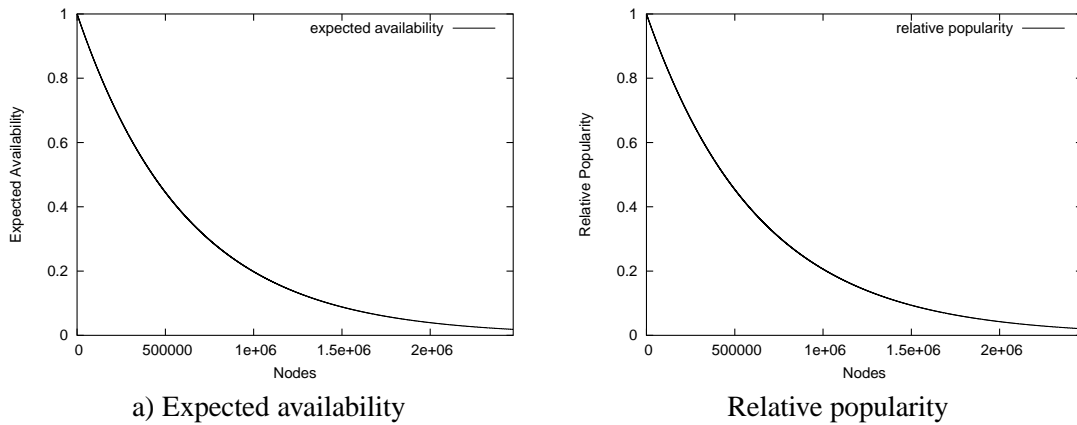
Looking at the correlation distribution, one can see that the peaks stemming from the well-behaving and the malicious selections move slightly closer to each other when the size is limited, but they are still separated enough to clearly identify the malicious ones.

Using  $n_{limit} = 250'000$  has its limits, too. If we perform the same experiment as above with in a system with 2'500'000 nodes, we get the results in figure 36.



**Figure 36:** Performance with a limited internal table size and 2'500'000 nodes in the system

Although most of the fully compromised anonymous tunnels can still be detected, comparing the correlation distributions in figure 36 show that distinguishing the two peaks when the size of the internal table is kept limited becomes more and difficult as the number of nodes in the system grows. The reason for this is illustrated in figure 37. It depicts the expected availability for each node according to formula 9 and the relative popularity of the nodes.

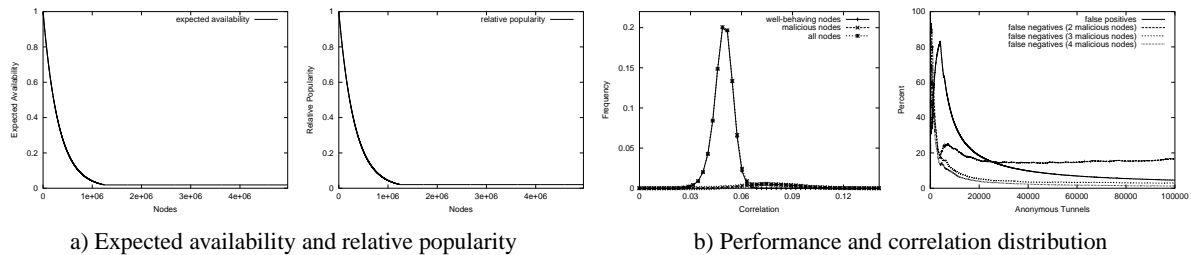


**Figure 37:** Expected availability and relative popularity

Looking at figure 37, we can see that the expected availability of the first 500'000 nodes is 50% or more, and all of them are at least half as popular as the most popular node. Analyzing this in more detail yields that the 250'000 most frequently selected nodes are responsible for only 55% of all occurrences of well-behaving nodes in selections and a little more than the top 500'000 nodes for 80%. It seems that this is where the choice of  $n_{limit} = 250'000$  reaches its limits. Increasing  $n_{limit}$  would of course help at the price of an increased time to process a newly arriving selection, but we nevertheless believe that setting  $n_{limit}$  to 250'000 is enough to cope with even very large real-world systems. In particular, assuming that the vast majority of all participating nodes are either only occasionally active or relatively unpopular, we can handle huge systems consisting of several million nodes.

In Figure 38, we illustrate the performance of a system consisting of 5'000'000 nodes. We assume that the majority of the participants are home users with a relatively poor network connectivity, which makes these nodes quite unpopular. We model this by assuming that 75% of all nodes are 50 times less popular than the most popular nodes. In addition, we assume that several of these home users use dial-up network connections, which means that their computers are not connected to the Internet most of the

time. We model this by setting the expected availability of 75% of the nodes about 50 times smaller than the expected availability of the most frequently available nodes.



**Figure 38:** Performance with a limited internal table size and 5'000'000 nodes in the system

Figure 38 shows that the system still works very well despite its many participating nodes and the relatively low limit  $n_{limit} = 250'000$  used to compute the maximum size of the internal table.

## 5 Related Work

We distinguish between approaches that use static set of MIXes and those that use a peer-to-peer based approach such as MorphMix.

### 5.1 Approaches Based on Static Sets of MIXes

The remailer network based on mixmasters [9] makes use of MIXes to send and receive e-mail messages anonymously. The sender selects a number of mixmasters and encrypts the message repeatedly for each mixmaster, starting with the last. The message is then sent via the selected mixmasters to the intended recipient. When a mixmaster receives an e-mail, it strips off the top layer of encryption which reveals the next mixmaster to use or the intended recipient. The sender can also include a reply block which allows the receiver to answer the sender's e-mail. The mixmaster network closely follows Chaum's original design of MIXes and makes use of uniform message length, replay detection, reordering, and batching of messages. As a result, the mixmaster network is quite resistant against traffic analysis attacks and offers a high level of anonymity.

A system to anonymize ISDN-telephony [18] via a MIX cascade is based on the idea that the subscribers connected to the same end-office build an anonymity group. The approach makes heavily use of the synchronized telephony system in the sense that all subscribers are always sending data to the end-office so that real phone calls cannot be distinguished from the dummy data. Although the idea can be realized very efficiently in the telephony world, it is questionable how well it is suited for the highly asynchronous Internet.

The Anonymizer [10] is a simple proxy-based service that offers anonymous Web browsing. The system works similar as a Web proxy in the sense that all data exchanged between the user's browser and the Web server are relayed by the Anonymizer. The advantages of the system are that it is simple and that the delay it introduces is relatively low compared to more sophisticated systems. The disadvantages are that the level of anonymity it offers is quite low and that the end-to-end relationship is not anonymous with regard to the Anonymizer itself.

Onion Routing [20] is a MIX network to enable anonymous use of near real-time Internet services. The system employs uniform message length and layered encryption of messages to complicate traffic analysis. The system delivered first results on anonymizing delay-sensitive Internet traffic via a MIX network, but went off-line in January 2000. One result of Onion Routing was that MIX networks are vulnerable to sophisticated traffic analysis attacks if no dummy traffic is used. The Onion Routing

analysis and visualizations pages<sup>1</sup> provide some interesting quantitative results that were collected during the operation of their prototype. A second generation system of Onion Routing which should offer much better protection has been underway as of June 2000 [26]. It should be noted that the U.S. government was awarded a patent for Onion Routing on July 24th, 2001. It is uncertain what impact this patent has.

The Freedom System [5] was a commercial service provided by Zeroknowledge Systems to browse the Web anonymously. Its approach is very similar to that of Onion Routing, and the mixing components are the *Anonymous Internet Proxies* (AIPs). In the first version of their system, a route employed always three AIPs, all packets in the system had the same length, and dummy traffic was used to further complicate traffic analysis. In the second version of their system, they reduced the number of AIPs in a route to two and got rid of dummy traffic or fixed-length packets. The system designers' argument is that the increased resistance against traffic analysis is not worth the bandwidth overhead. In addition, the AIPs do not really mix the traffic but forward the packets in a FIFO manner. As a result, Freedom is not very resistant against traffic analysis attacks [1]. The Freedom network has been shutdown as of October 22th, 2001, probably due to economical reasons [25]. However, since the Freedom network was operational for nearly two years, their experiences during the operation of such a service could be very interesting and helpful for the community.

Web MIXes is another promising project [3, 4] that wants to provide anonymous access to near real-time services in the Internet. They assume a very strong attacker model and use a MIX cascade to complicate traffic analysis. A prototype of their system is actually up and running<sup>2</sup> but does not yet provide the kind of resistance against attacks they are aiming at. However, our trials of their system provided acceptable performance for Web browsing.

There is more work going on on anonymity in the Internet. In November 2000, the NymIP effort [7] started to create a set of standardized protocols for pseudonymity and anonymity at the IP layer. Just recently, two Internet-drafts were published that describe anonymizing packet forwarders [16, 6]. Finally, work is being conducted to increase the reliability of MIX-based systems [12].

## 5.2 Approaches Based on Peer-To-Peer Technology

Crowds [21] works by collecting Web 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 of the crowd. If a user requests an URL, this request is forwarded randomly to another member in the crowd. Whenever a crowd member receives a request from another member, it makes a random choice to either forward the request to another crowd member (also chosen randomly) or submit it to the server the request is intended for. The reply from the server uses the same path back. If the crowd is large enough, then neither the other members nor the server nor any eavesdropper outside the crowd can tell which member in the crowd initiated the request and 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 of a crowd to link-encrypt the messages. Crowds has its weaknesses against local eavesdroppers (that monitor the traffic within the crowd) and collaborating members.

Tarzan [14, 13] 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 A maintains a bidirectional packet stream with a fixed number of other nodes (the *mimics* of A). Anonymous tunnels through A are only relayed via A's mimics, which implies that real data are always hidden in the packet streams between A 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

---

<sup>1</sup><http://www.onion-router.net/Analysis.html>, <http://www.onion-router.net/Vis.html>

<sup>2</sup><http://anon.inf.tu-dresden.de/index.html>

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 all other 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.

## 6 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 3.1, we argue that we have achieved them. Joining the system is easy, as 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.

MorphMix is resistant against collusion attacks because after having acquired enough knowledge, a node is able to identify anonymous tunnels containing several malicious nodes with very high probability. Although we cannot quantify how much protection MorphMix offers from traffic analysis attacks, we have reasonably argued that monitoring the whole system is extremely difficult due to the vast number of participating nodes. In particular, monitoring a specific user is difficult as the anonymous tunnels used by her are only short-lived and use each of them is relayed by completely different nodes. Acceptable end-to-end performance is achieved by quickly switching to another tunnel when one offers very poor performance or has stopped working completely.

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. Assuming a realistic scenario, we have shown that even with a few million nodes in the system, we can keep the time to process a newly arriving selection below 100 milliseconds in Java. Larger systems can be handled by increasing the size of the internal table, but at the cost that the time to process a new selection increases linearly with the number of nodes in the system.

Several open issues remain to be solved. Our next steps are to 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 10'000 nodes in only a few different subnets or regularly assign a different IP address within the same subnet to a node to change its identity from time to time.

Once we have solved that, we will work on the remaining details of MorphMix, design its protocol and implement the system.

## References

- [1] Adam Shostack Adam Back, Ian Goldberg. Freedom 2.1 Security Issues and Analysis. White Paper, [http://www.freedom.net/info/whitepapers/Freedom\\_Security2-1.pdf](http://www.freedom.net/info/whitepapers/Freedom_Security2-1.pdf), May 3 2001.
- [2] T. Berners-Lee, R. Fielding, and H. Frystyk. Hypertext Transfer Protocol – HTTP/1.0. RFC1945, 1996.

- [3] 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.
- [4] Oliver Berthold, Hannes Federrath, and Stefan Köpsell. Web MIXes: A System for Anonymous and Unobservable Internet access. In *Proceedings of the Workshop on Design Issues in Anonymity and Unobservability*, pages 101–115, Berkeley, CA, USA, July 25–26 2000.
- [5] 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](http://www.freedom.net/info/whitepapers/Freedom_System_2_Architecture.pdf), December 18 2000.
- [6] S. Bradner and H. Kung. Requirements for an Anonymizing Packet Forwarder. <http://www.ietf.org/internet-drafts/draft-bradner-annfwd-req-00.txt>, work in progress, November 2001.
- [7] Scott Bradner. The NymIP Effort. <http://nymip.velvet.com>.
- [8] David L. Chaum. Untraceable Electronic Mail, Return Addresses, and Digital Pseudonyms. *Communications of the ACM*, 24(2):84–88, February 1981.
- [9] Lance Cottrell. Mixmaster Software. <http://www.obscura.com/~loki>.
- [10] Lance Cottrell. The Anonymizer. <http://www.anonymizer.com>.
- [11] W. Diffie and M. E. Hellman. New directions in cryptography. *IEEE Transactions on Information Theory*, 22(6):644–654, November 1976.
- [12] 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*, Pittsburg, PA, USA, April 2001.
- [13] Michael J. Freedman. A Peer-to-Peer Anonymizing Network Layer. M.Eng Thesis, Massachusetts Institute of Technology, Cambridge, MA, USA, May 2002.
- [14] Michael J. Freedman, Emil Sit, Josh Cates, and Robert Morris. Introducing Tarzan, A Peer-to-Peer Anonymizing Network Layer. In *Proceedings of the 1st Intl. Workshop on Peer-to-Peer Systems (IPTPS 2002)*, Cambridge, MA, USA, March 7–8 2002.
- [15] R. Housely and W. Polk. Internet X.509 Public Key Infrastructure. RFC 2528, 1999.
- [16] H. Kung and S. Bradner. A Framework for an Anonymizing Packet Forwarder. <http://www.ietf.org/internet-drafts/draft-kung-annfwd-framework-00.txt>, work in progress, November 2001.
- [17] Andreas Pfitzmann and Marit Köhntopp. Anonymity, Unobservability, and Pseudonymity - A Proposal for Terminology; Draft v0.12. [http://www.koehtopp.de/marit/pub/anon/Anon\\_Terminology.pdf](http://www.koehtopp.de/marit/pub/anon/Anon_Terminology.pdf), June 17 2001.
- [18] Andreas Pfitzmann, Birgit Pfitzmann, and Michael Waidner. ISDN-MIXes: Untraceable Communication with Very Small Bandwidth Overhead. In *Kommunikation in verteilten Systemen*, 267, pages 451–463, 1991.
- [19] Sandro Rafaeli, Marc Rennhard, Laurent Mathy, Bernhard Plattner, and David Hutchison. An Architecture for Pseudonymous e-Commerce. In *Proceedings of the AISB'01 Symposium on Information Agents for Electronic Commerce*, pages 33–42, York, UK, March 21–24 2001.

- [20] 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.
- [21] 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.
- [22] 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.
- [23] 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)*, Pittsburgh, USA, June 10–12 2002.
- [24] 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.
- [25] Zeroknowledge Systems. Shutdown of Freedom Network. <http://www.freedom.net/prem.html>, October 2001.
- [26] 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.
- [27] 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.