Buses for Anonymous Message Delivery*

Amos Beimel Shlomi Dolev
Department of Computer Science
Ben-Gurion University of the Negev
Beer-Sheva 84105, Israel
beimel,dolev@cs.bgu.ac.il

April 11, 2002

Abstract

This work develops a novel approach to hide the senders and the receivers of messages. The intuition is taken from an everyday activity that hides the "communication pattern" – the public transportation system. To describe our protocols, busses are used as a metaphor: Busses, i.e., messages, are traveling on the network, each piece of information is allocated a seat within the bus. Routes are chosen and buses are scheduled to traverse these routes. Deterministic and randomized protocols are presented, the protocols differ in the number of buses in the system, the worst case traveling time, and the required buffer size in a "station." In particular, a protocol that is based on cluster partition of the network is presented; in this protocol there is one bus traversing each cluster. The clusters' size in the partition gives time and communication tradeoffs. One advantage of our protocols over previous works is that they are not based on statistical properties for the communication pattern. Another advantage is that they only require the processors in the communication network to be busy periodically.

Key Words. Anonymous communication, Privacy, Traffic analysis.

1 Introduction

Throughout the history encryption was used to hide the *contents* of transmitted data. The rapid growth in the use of the Internet only increased the necessity of encryption. However, encryption does not hide all the relevant information, for example, it does not hide the identity of the communicating parties. That is, it does not prevent traffic analysis. In this work we deal with the problem of anonymous communication – communication that does not disclose the identity of the sender and receiver.

*A preliminary version of this paper was published in *Proc. of the 2nd International Conference on FUN with Algorithms*, pages 1 – 13, 2001.

We develop a novel approach to hide the senders and the receivers of messages. The intuition is taken from an everyday activity that hides the "communication pattern" – the public transportation system. For example, a traveler that takes buses from one place to another remains anonymous, and it is hard to trace him. Metaphorically, we consider the pieces of information that senders send to receivers as passengers. There are "buses," i.e., messages, traveling on the network, and each piece of information is allocated a seat within a bus. The sender and receiver are modeled as bus stations. Our aim is to simulate this metaphor in the digital world while keeping the anonymity of the sender and receiver. In most of our protocols we also hide the information that a message is sent, that is, hide the number of passengers on each bus.

Previous work. One of the first works to consider the problem of hiding the communication pattern in the network is the work of Chaum [2] where the concept of a *mix* is introduced. A single processor in the network, called a mix, serves as a relay. Each processor p that wants to send a message M to a processor q encrypts M using q's public key to obtain M'. Then p encrypts the pair (M', q) using the public key of the mix. The mix decrypts the message and forwards M' to q. This scheme has been extended in [10, 11, 13, 14] where several mixes are used to cope with the possibility of compromising the single mix. For example, in the onion routing system [14] a proxy defines a route for a message through the routing network by rapping the message with a layered data-structure called an onion; the onion is passed through the routers as specified by the onion, each router which receives an onion peels of its layer, identifies the next hop in the route, and sends the peeled onion to the next router. The mix schemes operate under some statistical assumption on the pattern of communication. If a single message is sent then an adversary that monitors the communication channels can observe the sender and the receiver of the particular message. Another example for a problematic case is when all the processors send a message to the same destination – in this case the identity of the receiver is revealed. A discussion on other mix-like systems can be found in [14].

An approach based on "xor-trees" has been presented in [6]. The scheme presented in [6] fits long communication sessions during which the data exclusive-ored with pseudo-random bits (that cancel each other) is transfered towards the root which in turn broadcast the arriving information to the nodes in the tree. The solution presented in [6] is an extension of the DC-net approach suggested in [3].

Our contribution. We present deterministic and randomized protocols for anonymous message delivery based on the buses metaphor. The protocols differ in the number of buses in the system, the worst case traveling time, and the required buffer size in a "station." Our first solution uses a single bus that traverses an Euler tour. The traveling time in this protocol is $O(n)$. Our second solution is a full communication solution for which two buses traverse each link in opposite directions. The traveling time in this protocol is the distance between the sender and receiver, that is, the protocol achieves optimal time. Our third solution is based on cluster partition of the network; in this protocol there is one bus traversing each cluster. The clusters' size in the partition gives time and communication tradeoffs.

Our solutions do not rely on statistical properties for the communication pattern. In our solution, unlike the solution presented in [6], the processors in the communication network are busy in the transmission only periodically. That is, a processor is busy only when a bus arrives at the processor. Moreover, there is no need to store information (such as the result of xoring arriving bits) in memory between bus arrivals; thus, our protocols are more suitable for fault tolerant environments. For example, the scheme may be a base for a robust anonymous message delivery by retransmitting a new *bus* upon a time-out.

Let us note two additional important properties of our scheme. First note that in our protocols the buses traverse the network in fixed routes and fixed schedule, thus the adversary cannot learn whether there is any communication between the processors or not. In addition, our scheme can cope with an adversary that monitors any number of processors.

We also extend the scheme to cope with three extensions of the model: (1) protocols that enable anonymous broadcast and multicast, (2) protocols which work even if the topology of the network is unknown, and (3) protocols which tolerate Byzantine processors.

Organization. The rest of the paper is organized as follows. The problem statement appears in Section 2. Two simple solutions that achieve minimum communication and minimum time, respectively, are presented in Section 3. A solution that introduces a tradeoff between time and communication is presented in Section 4. Lower bounds on the possible tradeoffs between time and communication are proved in Section 5. Solutions which cope with extensions of the model appear in Section 6.

2 The System and Threat Models

We consider a network of n processors, denoted $p_1, \ldots, p_n$, connected by m communication links. We use the communication graph $G(V, E)$ to represent our network, V is the set of processors and E is the set of communication links connecting the processors (that is $n = |V|$ and $m = |E|$). We assume that G is connected. Processors communicate by sending and receiving messages. The system is synchronous – there is a common global *pulse* (possibly implemented by synchronized distributed clocks) that triggers (whenever the clock reaches an integer value) the processors to send messages; messages sent in a certain pulse arrive at the neighboring processor before the next pulse.

Some processor p_i, called *sender*, may decide to communicate with another (not necessarily neighboring) processor p_j, called *receiver*. Informally, our objective is to hide the fact that p_i communicates with p_j. That is, we want to hide the identities of p_i and p_j. See below for a formal definition. Furthermore, some of our protocols even hide the fact that a message was sent. A protocol that achieves these goals is called an *anonymous message delivery protocol.* We note that the vast majority of known cryptographic techniques focus on hiding the *contents* of the transmitted data, but not the fact that data has been transmitted.

We consider two types of adversaries *listening adversary* and *Byzantine adversary.* The listening adversary can monitor *all* the communication links and also monitor the internal contents of some processors in the network. The adversary is non-adaptive: Before the execution of the protocol the adversary chooses a set of processors $C \subseteq \{p_1, \ldots, p_n\}$ (we do not limit the size of the set). Later a pair of parties (or some pairs) execute the anonymous message delivery protocol. At the end of the executions the adversary should not know if p_i sent a message to p_j for every $p_i, p_j \notin C$.[1] This adversary is honest-but-curious, i.e., it cannot change any messages, delete messages, add any messages, or change the state of any processor. We next formally define anonymous message delivery protocols. For this definition, we recall the definition of indistinguishable distributions [9, 16].

Definition 2.1 (Indistinguishability) *Two sequences of probability distributions, $\{V_k\}_{k=1}^{\infty}$ and $\{W_k\}_{k=1}^{\infty}$, are* polynomial-time indistinguishable *if for every probabilistic polynomial-time Turing Machine M, every integer $c \geq 1$ and for every sufficiently large k,*

$$\left| \Pr_{v \in V_k} \left[M\left(v, 1^k\right) = 1 \right] - \Pr_{w \in W_k} \left[M\left(w, 1^k\right) = 1 \right] \right| \leq \frac{1}{k^c}.$$

The protocols we define have a security parameter k which measures the length of the keys in the cryptographic primitives they use (see Section 2.1 for description of these primitives). Roughly speaking, the requirement is that a Turing Machine that runs in time polynomial in k cannot know who are the sender and receiver. The view of the listening adversary controlling a set C of processors after an execution of a message delivery protocol in which p_i sends a message to p_j is denoted by the random variable $\mathrm{VIEW}_C^k(i,j)$ where k is the security parameter. This view contains all messages exchanged in the network and the local information known to processors in C, i.e., the random inputs they used during the execution, the state of the processors, and the secret keys they know.

Definition 2.2 *We say that the protocol is an* anonymous message delivery protocol *if:*

Correctness. *If p_i sends a message M to p_j then p_j receives the message.*

Anonymity. *For every $C \subseteq \{p_1, \ldots, p_n\}$ and every $i_1, i_2, j_1, j_2 \notin C$ the sequences of random variables $\left\{\mathrm{VIEW}_C^k(i_1, j_1)\right\}_{k=1}^{\infty}$ and $\left\{\mathrm{VIEW}_C^k(i_2, j_2)\right\}_{k=1}^{\infty}$ are indistinguishable.*

The *Byzantine adversary* is more powerful than the listening adversary. Like the listening adversary, the Byzantine adversary can monitor the communication links and the internal contents of the processors of the network. In addition, for some parameter t, it can control up to t processors in the network. These processors can insert messages, delete messages, or arbitrarily change messages that they receive (before forwarding the messages). That is, these processors can deviate from the pre-defined protocol. Again we assume that the adversary is non-adaptive. We do not give a formal definition of an anonymous message delivery protocol in the presence of a Byzantine adversary since it is quite complicated. The definition is similar to the definition of secure function evaluation in the presence of a Byzantine (malicious) adversary.

We evaluate a protocol by its *time complexity*, its *communication complexity*, and its *buffer complexity*: the time complexity is the worst case time required to transmit a message

[1] If the adversary monitors the internal contents of the sender or the receiver, then it can identify the sender and the receiver.

from a sender to a receiver, the communication complexity is the maximal number of messages that are sent simultaneously by the processors in the network (however, these messages can be long), and the buffer complexity is the buffer size required for each processor to store incoming and outgoing messages in each time step. In our protocols the buffer complexity is the number of seats in the buses that arrive simultaneously to a processor.

2.1 Cryptographic Primitives

The first cryptographic primitive that we use is *encryption* which guarantees the secrecy of messages. That is, a sender can send an encrypted message such that only the intended recipient can decrypt it. We will require *semantic security* – the encryption is randomized and an eavesdropper cannot distinguish in polynomial time between encryptions of any pair of messages. See, e.g., [7] for formal definitions. We consider two types of encryption:

Symmetric key encryption. Both sender and receiver have a common secret key, which is used for both encryption and decryption.

Public key encryption. The receiver has a secret private key which it uses for decryption. Furthermore, the receiver publishes a public key which is used for encryption by any sender; an eavesdropper cannot distinguish between encryptions of messages even if it has the public key.

Typical symmetric key encryption schemes are faster than public key encryption schemes; however they require every pair of processors in the network to have a common secret key.

Authentication. The second cryptographic primitive that we use is *authentication* which guarantees that if a sender sends a message to a receiver and a third party alters this message, then with high probability the receiver can detect this fact. Again we can consider two types: (1) symmetric key authentication in which the sender uses the common key to authenticate a message and the receiver uses the common key to verify the authenticity of the message, and (2) public key authentication, known as signatures, in which the sender uses its private key to sign a message and the receiver uses the public key to verify the validity of the signature. See, e.g., [8] for formal definitions of authentication and signatures.

3 Simple Solutions

In this section we present two simple protocols, one with optimal communication complexity and another with optimal time complexity. In Section 4 we will generalize the ideas of these protocols, and present protocols that exhibit tradeoffs between time and communication. In all our protocols we metaphorically view each message as a *bus*. The protocols vary according to the number of buses in the system, and the way they travel in the communication graph.

3.1 Communication Optimal Protocol

We start with a solution with message complexity 1, i.e., in each time unit only one processor sends a message to one other processor. Using our metaphor, there is only one bus traveling in the system. We next define how the bus travels in the communication graph. First, fix any spanning tree in the graph. Next, use an Euler tour (that is, a DFS tour) of the spanning tree to define a ring. The bus is rotating through the ring, and has n^2 *seats*. Seat $s_{i,j}$ is used to communicate an encrypted message from processor p_i to p_j; this message is encrypted either using the symmetric key of p_i and p_j, or using the public key of p_j (depending which encryption infra-structure exists). Each time the bus gets to processor p_i it changes each message in the *row* of seats $s_{i,j}$ either to an encryption of a message it wants to send to p_j, or to some detectable garbage which is then encrypted for p_j. Furthermore, p_i checks what messages were sent to it, by decrypting the n messages located in the i-th *column* and ignoring the ones containing garbage.

By the semantic security of the encryption, a listening adversary cannot tell whether a seat contains garbage or a real message, i.e., it cannot tell if two processors are communicating. Next, we state the communication and time complexities of our solution. The communication complexity of this solution is optimal – there is single bus. However, the time complexity is quite bad: it can take at most $2n - 1$ time units until the bus reaches the sender, and at most $2n - 1$ additional time units until the bus reaches the destination. The buffer complexity of this protocol is n^2. We summarize the properties of this protocol below.

Theorem 3.1 *There is an anonymous message delivery protocol with communication complexity* 1*, time complexity* $O(n)$*, and buffer complexity* $O(n^2)$*.*

We emphasize again that since there is one bus, most of the time each processor is not involved in executing this protocol and does not need to store any information between two visits of the bus. Furthermore, by Theorem 5.2, the time complexity in any protocol with communication complexity 1 is $\Omega(n)$.

3.1.1 Reducing the Number of Seats

In this section we present a protocol that reduces the number of seats in each bus assuming that not too many messages are sent simultaneously. We modify the above protocol where instead of assigning a seat for any source/destination pair, the sender writes its message in a randomly chosen seat (deleting the previous contents of the seat). However, the sender wants to hide the fact that it wrote a message in some seat/seats, thus it changes the contents of all the seats in the bus. To achieve this goal, the sender encrypts the message using the public keys, in reverse order, of all the processors in the Euler tour between the sender and receiver. When the bus gets to some processor, it replaces the contents of each seat by the decryption of the previous contents under its private key. Next, if any message makes sense, then the processor knows that this is a message sent to it, and it changes it to a random contents. Recall that we use a semantically-secure public-key encryption; such encryption scheme must be probabilistic, and the length of the nested encryption, that is, after the multiple encryptions, is $O(n)$.[2] The sender appends dummy blocks to the encryption such that its length does not leak information on the intended receiver. For more details on semantically-secure encryption, see e.g., [7]).

The buffer complexity in the protocol is $O(n)$ times the size of the bus. To determine the size of the bus that may serve well under this policy we use the so-called birthday problem (or birthday paradox). As an example, with probability $1/2$ in a group of 23 random people there will be two people with the same birthday. More generally,

Claim 3.2 *Suppose s balls are randomly and independently assigned to r bins. The probability that all balls fall into distinct bins is $\approx e^{-s(s-1)/2r}$.*

Assume that we have an upper bound s on the number of messages that will be sent anonymously. Thus, if we take the size of the bus to be $r = O(s^2)$ then the probability that two processors will randomly choose the same seat is less than $1/4$. If we take the size of the bus to be $r = O(ks^2)$, for some security parameter k, then the probability of a collision drops to $1 - e^{-1/k} \approx 1 - (1 - 1/k) = 1/k$.

Of course, if there is a collision then the first message gets lost. A possible way to overcome this problem is that the recipient sends an acknowledgment to the sender using the same seat. If the sender does not get the message, then the sender resends the message. The expected number of times that a message will be sent is less than 2 even if the number of seats is $r = O(s^2)$.[3]

Theorem 3.3 *Assume that there is some upper bound s on the number of anonymous messages that are sent simultaneously. There is an anonymous message delivery protocol with communication complexity 1, expected time complexity $O(n)$, and buffer complexity $O(ns^2)$.*

The above protocol enables to send an anonymous-sender message, that is, a message in which the sender keeps its anonymity from the receiver (simply by not mentioning the originator of a message). Now, if a sender p_i sees that it resends a message many times, then p_i can decide to double the size of the bus. However, p_i does not want to reveal that it is trying to send a message, thus it can send an anonymous-sender message to another (random or fixed) processor to double the size of the bus. Similarly, a processor that receives acknowledgments for several messages in a row can send an anonymous message to reduce the size of the bus.

Another way to reduce the number of seats is to assume that each time the bus gets to p_i it will send only one message.[4] In this case, we can use a bus with only n seats: Each processor has a single seat s_i in the bus that can be used for sending a message to another processor in the ring. The message M is encrypted by the sender in a way that ensures that only the receiver can decrypt M. That is, when the bus gets to a processor p_j it tries to decrypt the messages in the $n - 1$ seats s_i, where $i \neq j$, and receives the messages that it can verify their authenticity.

[2] Here we ignore the security parameter.

[3] The disadvantage of this protocol is that when a sender does not get the acknowledgment then it knows that someone else is also sending messages.

[4] Alternatively, if p_i has more than one processor with whom it wants to communicate, then it will use a buffer to store these messages; this will increase the delivery time to $O(n^2)$.

3.2 Full Communication – Time Optimal Protocol

We next present a protocol with optimal time complexity, however with bad communication complexity. In this protocol two buses travel through every link – a bus in each direction. The nodes transfer seats from one bus to another according to the shortest path criteria. A seat $s_{i,j}$ that arrives at a node p_k is assigned to a bus that traverses the link attached to p_k that is on a shortest path to p_j. The seats that are transfered use the routing information, and may be transfered together with the routing messages that are repeatedly exchanged. That is, the communication in this protocol is "swallowed" by the communication of the routing-update protocol.

As in the previous protocol all messages are encrypted using the key of the receiver before they are assigned to seats, and encrypted garbage messages are sent if there is no real message. Thus, anonymity is guaranteed. Next we state the communication and time complexities of this protocol. The communication complexity of this protocol is the number of buses, i.e., $2m$ (where m is the number of edges in the graph). This protocol has optimal time for message arrival, which is the number of links in the shortest path between the receiver and sender. The buffer complexity of a node is the number of shortest paths that contain this node. This number can be small or big depending on the communication graph. For example, if the graph is a complete graph, each bus contains one seat, and the buffer complexity of a node is the number of its neighbors, i.e., $n-1$, however, the number of buses is $O(n^2)$. The other extreme is a star, where the buffer complexity of the center is $O(n^2)$ and the number of buses is $O(n)$.

Theorem 3.4 *There is an anonymous message delivery protocol with communication complexity $2m$ and buffer complexity at most $O(n^2)$. The time complexity between two nodes is the distance between the nodes in the communication graph.*

4 Time and Communication Tradeoff

In this section we will examine more sophisticated protocols that can be tuned up to trade time and communication. The first observation is that the full communication protocol presented in Section 3.2 already presents tradeoffs between time and communication: the protocol can use any connected spanning sub-graph of the communication graph with two buses on each edge of the subgraph. This reduces the communication complexity but might increase the time complexity since the distance between two nodes in the sub-graph might be bigger. To obtain the minimum number of buses, the protocol uses a spanning tree; in this case the communication complexity is $O(n)$.

We next present protocols which reduce the number of buses to less than n. In these protocols we divide the graph into clusters and construct bus routes within each cluster. For concreteness, we choose specific partitions to clusters that are based on [5], however similar partitions can be used as well (see the related work in [5]).

The partition scheme of [5] uses a spanning tree of the communication graph, and partitions its nodes and edges to clusters. One way to partition the tree is a node partition which results in clusters with at least x nodes and no more than δx nodes, where x can be chosen to be any value in the range $1, \ldots, n$ and δ is the maximum degree of a node in the tree. In this partition neighboring clusters are connected by a single link. The partition scheme that we will use is edge partition, that is, each edge in contained in exactly one cluster. In this case each cluster contains at least x edges and no more than $3x$ edges, where, again, x can be chosen to be any value in the range $1, \ldots, n$. (In fact at most one cluster is of size $3x$ and all the rest are of at most $2x$.) Each cluster is a connected sub-graph of the spanning tree, i.e., it is a tree that contains $O(x)$ nodes. In this partition two neighboring clusters are connected by a single node.

We now roughly describe the edge partition scheme of [5]. A rooted spanning tree is constructed and each node p is marked by M_p, the number of edges in its subtree. In each iteration a node with $M_p \geq x$, such that for all p's children q it holds that $M_q < x$, is chosen. Then a subset of the subtrees rooted at p's children are selected such that the total number of the edges in these subtrees is greater than x but not exceeding $2x$. These trees form a cluster, that is removed from the tree. Now, the numbers M_p recalculated for the remaining tree, and the scheme proceeds to the next iteration. Note that if the number of edges in the tree is less than $3x$ then it may not be possible to partition the last remaining tree into a cluster of x to $2x$ edges. For example, a root with three outgoing edges for which the subtree rooted at each of them is of size exactly $x - 1$ cannot be partitioned as we require – hence we allow the last cluster to include $3x$ edges.

Once the network is partitioned to clusters, we have one bus in each cluster which performs an Euler tour on the spanning tree of the cluster. There are at most $\lceil n/x \rceil$ clusters in the graph, thus the number of buses, i.e., the communication complexity, is no more than $\lceil n/x \rceil$. If a message is sent from a node in one cluster to a node in another cluster then this message should move from one bus to another until it reaches the cluster of the receiver. That is, when a bus reaches a node that is part of more than one cluster (recall that we use an edge partition), then seats are transfered from one bus to another. The bus in $Cluster_\ell$ has a seat $s_{i,j}$ for every p_i and p_j such that the simple path connecting them in the spanning tree passes through an edge of $Cluster_\ell$. We next analyze the buffer complexity: For a given node and a given seat $s_{i,j}$, there can be at most two clusters containing the node such that the path from p_i to p_j in the spanning tree uses an edge from the cluster. Thus, the buffer size of each node is at most twice the number of simple paths in the tree passing through the node. This number is at most $O(n^2)$. Since the messages are encrypted using a semantically secure encryption, the anonymity is guaranteed.

4.1 Bus Scheduling

We would like to minimize the time required for a message to arrive to its destination. To achieve this goal, buses in clusters with a common node, should reach the common node simultaneously in order to transfer seats. We show how to schedule the buses to satisfy this condition. Recall that we consider a synchronous settings, where the bus traverses an edge in a single time unit. Furthermore, we use the fact that clusters have similar sizes. Let us first consider an ideal case, where the clusters have identical size. In this case, we can start with an arbitrary cluster, schedule its bus, and whenever the bus reaches a node shared with another cluster, we start scheduling the bus of the neighboring cluster. Since we consider a spanning tree then there are no cycles and this scheduling is possible.

If the clusters have different number of nodes, we first schedule the bus in a cluster $Cluster_\ell$ with the maximum number of edges $m_{\max}$. Recall that an Euler tour in this cluster will take $2m_{\max}$ time units. Then whenever the bus reaches a node that is part of other clusters, the buses of the other clusters are scheduled. It is possible that a neighboring cluster $Cluster_j$ has $m' < m_{\max}$ nodes, in such a case the bus of $Cluster_j$ will wait, $O(m_{\max} - m')$ time units, for the bus of $Cluster_\ell$, whenever it reaches the node that is common to $Cluster_\ell$ and $Cluster_j$. The procedure continues in a fashion similar to the case of identical size clusters.

We next analyze the time complexity of this protocol. If the distance between node p_i and node p_j in the spanning tree is d, then the path can pass through at most d clusters, and in each cluster it would take less than $2m_{\max}$ steps until the message would pass to the next cluster. Thus, the delivery time from p_i to p_j is $O(dx)$ (since $m_{\max} < 3x$, where x is the parameter chosen in the edge partition scheme). In the worst case the message will pass through each edge of the spanning tree at most twice and the delivery time would be $O(n)$.

Theorem 4.1 *For every x, where $1 \leq x \leq n$, there is an anonymous message delivery protocol with communication complexity $O(n/x)$, buffer complexity $O(n^2)$, and time complexity between two nodes is $O(\min(dx, n))$, where d is the distance between the nodes in the spanning tree.*

Example 4.2 Consider a complete binary tree with a "natural" partition into clusters. More precisely, consider a complete binary tree whose height is ℓa for some parameters ℓ and a, and its number of nodes is $n = 2^{la+1} - 1$. We partition the tree into clusters of size $x \stackrel{\text{def}}{=} 2^{a+1} - 1$ where each cluster is a complete binary tree of depth a. The distance d between two nodes in this case is at most $2 \log n$. However, the upper-bound of Theorem 4.1 is too pessimistic for this case. Observe that any simple path in the tree passes through at most $2 \log n / \log x$ clusters, thus the delivery time is $O(x \log n / \log x)$ and the message complexity is $(n-1)/(x-1)$. See example in Figure 1.

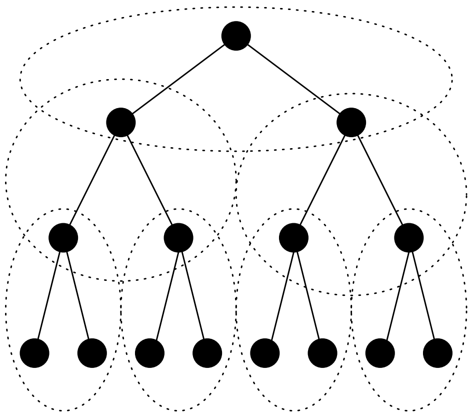

Figure 1: A tree of height 3 partitioned into 7 clusters of height $a = 1$.

4.2 Reducing the Number of Seats

We can reduce the number of seats in a bus, i.e., reduce the buffer complexity. We use a bus with $O(n^2/x^2)$ seats, a seat $s_{k,\ell}$ for a message that should be transfered from the k'th cluster to the ℓ'th cluster. In this case only one message can be sent at a time from a particular cluster to another cluster. It is possible that more than one processor in $Cluster_k$ will try to transmit a message to $Cluster_\ell$. We use a probabilistic approach, where each processor in $Cluster_k$ that would like to send a message to $Cluster_\ell$ uses a random function to decide whether to overwrite the seat $s_{k,\ell}$. To ensure that overwrites are not observed each message is changed at every node. To do so, every message is encrypted in a nested fashion, using all the keys of the processors in the route to the bus exchange node.

5 Lower Bounds

In this section we prove lower bounds on the time/communication tradeoffs. As a warm-up we start with the simple case where there is one bus traversing the communication tree according to some Euler tour. This tour, whose length is $O(n)$, traverses each leaf of the tree once and there are at least two leaves. Thus, for any two leaves u and v the distance between u and v or v and u in the tour is at least $n/2$, and it takes at least $n/2$ time units to send a message from u to v or from v to u. The next lemma generalized the above simple scenario. It considers a protocol where in each time step only one processors sends a message. The order of the processors sending the messages can be arbitrary, it may change in time, or even be randomized. In this case we consider a very long execution of the protocol, where processors exchange many messages. We measure the expected delivery time from p_i to p_j, where the expectation is taken over the many times that p_i sends a message to p_j.

Lemma 5.1 *In any protocol with communication complexity* 1*, there are two nodes in the graph such that the expected delivery time from one node to the second is* $\Omega(n)$*.*

Proof: A necessary condition for transmitting a message from a node u to a node v is that u sends some message on one of its outgoing edges. In each time unit there is at most one node sending a message. For any t, consider the sequence of nodes that send messages in the first t time units. (We do not assume anything about this sequence other than that it contains at most t nodes.) There is at least one node u that appears at most t/n times in this sequence. In other words, the *expected* distance from two occurrences of u in the sequence is $\Omega(n)$. Fix such u and pick any node v. Assume that u wants to send a message to v one time unit after each time that it appears in the sequence. It takes $\Omega(n)$ time units for u to send a message to v.

The above proof does not use the anonymity requirement of the delivery protocol, but only relies on the message complexity. There is one delicate issue that we should elaborate. By the assumptions on the order of sending message, this order might depend on the transmitting parties (if we use the anonymity requirement then this assumption might be reasonable). The simplest way to get rid of this problem is to fix a vertex v in advanced, and assume that each other vertex wants to transmit a message to v one time unit after each time that it appears in the sequence. ■

Note that every protocol with communication complexity c and time complexity t can be transformed into a protocol with communication complexity 1 and time complexity tc (since we consider a synchronous system). Thus,

Theorem 5.2 *In any protocol with communication complexity c, there are two nodes in the graph such that the expected delivery time from one node to the second is $\Omega(n/c)$.*

The above theorem implies that the tradeoff in Theorem 4.1 cannot be improved by a factor bigger than $O(d)$ where d is the distance between the two nodes in the spanning tree. We next show that if we consider the "natural" partition of the complete binary tree into clusters described in Remark 4.2 then we obtain message complexity n/x and time complexity $\Theta(x \log n / \log x)$. The upper-bound is shown in Remark 4.2. We next show that this upper bound is tight for this partition. To prove this claim consider an Euler tour in a complete binary tree with x nodes starting from the root, and let v be the first leaf visited in the tour. The distance in the tour between v and the root is $\Omega(x)$. Now we consider the complete binary tree, and define a sequence of $\log n / \log x$ nodes $v_0, v_1, \ldots, v_{\log n / \log x}$, where v_0 is the root of the tree, and v_i is a leaf in the cluster of v_{i-1} whose distance from v_{i-1} in the Euler tour of the cluster is $\Omega(x)$. Thus, the delivery time of a message from $v_{\log n / \log x}$ to the root is $\Omega(x \log n / \log x)$ no matter how the buses are scheduled.

6 Extensions

In this section we show how simple modifications to the idea of the buses can cope with three extensions to the model. The first extension is anonymous multicast and broadcast, the second is when the topology of the communication graph is unknown, and the third is to a Byzantine adversary, that is, an adversary that can cause processors to behave maliciously.

6.1 Anonymous Multicast and Broadcast

In this section we discuss informally how to anonymously multicast and broadcast a message. Anonymous broadcast enables a sender to broadcast a message to all processors without revealing its identity. To enable anonymous broadcast the sender only needs to send an anonymous-sender message to some (fixed or randomized) receiver p_j, that is, a message in which the sender keeps its anonymity from the receiver. This message will simply say "broadcast message M to all processors." Processor p_j uses any (non-anonymous) broadcast protocol to broadcast M. The protocol of Theorem 3.3 enables anonymous-sender messages hence enables anonymous broadcast. Furthermore, in all our protocols where we allocate a seat $s_{i,j}$ for sending a message from p_i to p_j (e.g., the protocol of Theorem 3.1) we can add a seat $s_{*,j}$ meaning that some anonymous processor wants to send a message to p_j, in this case the sender uses the nested encryption method described in Section 3.1.1 to hide the fact that it changes the content of a seat. If there are not to many anonymous broadcasts sent simultaneously and the sender selects a random p_j then this solution is efficient.

Multicast enables a sender to send a message to some subset D of processors. We consider three variants of anonymous multicast: (1) keeping the anonymity of the sender, (2) keeping the anonymity of the recipients, (3) keeping the anonymity of both the sender and the recipients. Anonymous-sender multicast reduces to sending an anonymous-sender message to a single processor in the multicast set saying "multicast message M to the processors in D." Anonymous-recipients multicast reduces to independently sending the message M anonymously to each processor in D. This can be done without any over-head in all our protocols where we allocate a seat $s_{i,j}$ for each pair of processors. Finally, anonymous-sender, anonymous-recipients multicast can be achieved by independently sending the message M to each processor in D using an anonymous-sender protocol.

6.2 Unknown Topology

We consider the scenario where the processors in the network do not know the topology of the network (for example, the network can change periodically). The solution we propose for this problems is to use a random walk on the communication graph. More precisely, there is one bus traversing the graph, and in each step the processor holding the bus chooses uniformly one of its neighbors, and sends the bus to the chosen neighbor. Aleliunas et al. [1] proved that the expected time of a random walk that visits all the nodes of an undirected graph with n nodes and m edges is $O(nm)$. Thus, the expected delivery time of a message using a random walk (in an unknown graph) is $O(nm)$. This bound on the delivery time is tight for some graphs, e.g., the so called lollipop graph. However, it is too pessimistic for some graphs, e.g., for a clique the expected delivery time is $O(n \log n)$ (and not $O(n^3)$).

Theorem 6.1 *There is an anonymous message delivery protocol in a network whose topology is unknown with communication complexity* 1*, expected time complexity* $O(nm)$*, and buffer complexity* $O(n^2)$*.*

6.3 Byzantine Adversary

We now turn to the case in which processors are Byzantine, that is, they may try to add/delete or change messages in a malicious way. First note that the communication graph must be $t + 1$ connected in order to tolerate t faults. Otherwise, there is a cut of t or less Byzantine processors that can partition the graph into two isolated connected components. We therefore assume that the communication graph is $t + 1$ connected, thus, by Menger's theorem, for every two nodes there are $t + 1$ paths connecting them such that there is no internal node common to two of these paths. For every pair of processors we fix such $t + 1$ disjoint paths. We describe a protocol in which there are two buses on each link, one in each direction. When p_i wants to anonymously send a message to p_j, then p_i authenticates this message using a private key common to p_i and p_j. Processor p_i sends the message over the $t + 1$ fixed disjoint paths, therefore the message will reach the destination through at least one path with no Byzantine processor. This ensures that a Byzantine processor can not generate/change a message originating from some sender in a way that is not identified by the receiver. Thus, the Byzantine processor can only drop messages. To achieve anonymity we use the mechanism of the full communication protocol described in Section 3.2. The number of seats in a bus equals to the number of paths that use this link in the bus traveling direction. The time complexity from p_i to p_j in this protocol is the length of the longest path amongst the $t+1$ disjoint paths from p_i to p_j. In the worst case this can be n. We summarize the properties of the above protocol below.

Theorem 6.2 *Assume that the communication network is $t+1$ connected. There is an anonymous message delivery protocol against a Byzantine adversary that controls at most t processors with communication complexity $2m$, time complexity $O(n)$, and buffer complexity $O(n^2)$.*

We next discuss how to reduce the number of buses. Given a communication graph that is at least $t+1$ connected, we will find a spanning sub-graph that is $t+1$ connected and contains fewer edges. Finding a $t+1$-connected spanning subgraphs that has the minimum number of edges is NP-hard. However, there are good approximation algorithms for this problem. A recent result [4] describe an efficient algorithm that returns a graph whose number of edges is no more than $1+1/(t+2)$ times the optimal number of edges. In particular, the number of edges is no more than $(t+1)n$. This, however, might increase the delivery time since the length of the $t+1$ disjoint paths might be longer. We summarize the properties of the above protocol below.

Theorem 6.3 *Assume that the communication network is $t+1$ connected. There is an anonymous message delivery protocol against a Byzantine adversary that controls at most t processors with communication complexity $2(t+1)n$, time complexity $O(n)$, and buffer complexity $O(n^2)$.*

References

[1] R. Aleliunas, R. M. Karp, R. J. Lipton, L. Lovász, and C. Rackoff. Random walks, universal traversal sequences, and the complexity of maze problems. In *Proc. of the 11th Annu. ACM Symp. on the Theory of Computing*, pp. 218–223, 1979.

[2] D. Chaum. Untraceable electronic mail, return addresses, and digital pseudonyms. *Communication of the ACM*, vol. 24, no. 2, pp. 84–88, 1981.

[3] D. Chaum. The dining cryptographers problem: unconditional sender and recipient untraceability. *Journal of Cryptology*, vol. 1, no. 1, pp. 65–75, 1988.

[4] J. Cheriyan and R. Thurimella. Approximating minimum-size k-connected spanning subgraphs via matching. *SIAM J. on Computing*, vol. 30, no. 2, pp. 528–560, 2000.

[5] S. Dolev, E. Kranakis, D. Krizanc, and D. Peleg. Bubbles: adaptive routing scheme for high-speed dynamic networks. *SIAM J. on Computing,* vol. 29, no. 3, pp. 804–833, 1999.

[6] S. Dolev and R. Ostrovsky. Xor-trees for efficient anonymous multicast and reception. *ACM Transactions on Information and System Security*, vol. 3, no. 2, pp. 63–84, 2000.

[7] O. Goldreich. Fragments on a chapter on Encryption Schemes. Extracts from *Foundations of Cryptography* vol. 2 (in preparation). 2001. Available from http://www.wisdom.weizmann.ac.il/∼oded/foc-vol2.html.

[8] O. Goldreich. Fragments on a chapter on Digital Signature and Message Authentication. Extracts from *Foundations of Cryptography* vol. 2 (in preparation). 2001. Available from http://www.wisdom.weizmann.ac.il/∼oded/foc-vol2.html.

[9] S. Goldwasser and S. Micali. Probabilistic encryption. *J. of Computer and System Sciences*, vol. 28, no. 21, pp. 270–299, 1984.

[10] A. Pfitzmann. How to implement ISDNs without user observability – some remarks. TR 14/85, Fakultat fur Informatik, Universitat Karlsruhe, 1985.

[11] A. Pfitzmann, B. Pfitzmann, and M. Waidner. ISDN-MIXes – Untraceable communication with very small bandwidth overhead. In *Proc. Kommunikation in verteilten Systemen*, pp. 451–463, 1991.

[12] T. Rabin and M. Ben-Or. Verifiable secret sharing and multiparty protocols with honest majority. In *Proc. of the 21st ACM Symp. on the Theory of Computing*, pp. 73–85, 1989.

[13] C. Rackoff and D. Simon. Cryptographic defense against traffic analysis. In *Proc. of the 25th Annu. ACM Symp. on the Theory of Computing*, pp. 672–681, 1993.

[14] M. G. Reed, P. F. Syverson, and D. M. Goldschlag. Anonymous Connections and Onion Routing. *IEEE Journal on Selected Areas in Communication*, vol. 16, no. 4, pp. 482–494, 1998.

[15] M. Waidner and B. Pfitzmann. The dining cryptographers in the disco: unconditional sender and recipient untraceability with computationally secure serviceability. In *Advances in Cryptology – EUROCRYPT 89*, vol. 434 of LNCS, pp. 690, Springer-Verlag, 1990.

[16] A. C. Yao. Protocols for secure computations. In *Proc. of the 23th Annu. IEEE Symp. on Foundations of Computer Science*, pp. 160–164, 1982.