# Distributed Broadcasting and Mapping Protocols in Directed Anonymous Networks[*]

Michael Langberg [†] Moshe Schwartz [‡] Jehoshua Bruck [§]

**Abstract**

We initiate the study of distributed protocols over directed anonymous networks that are not necessarily strongly connected. In such networks, nodes are aware only of their incoming and outgoing edges, have no unique identity, and have no knowledge of the network topology or even bounds on its parameters, like the number of nodes or the network diameter. Anonymous networks are of interest in various settings such as wireless ad-hoc networks and peer to peer networks. Our goal is to create distributed protocols that reduce the uncertainty by distributing the knowledge of the network topology to all the nodes.

We consider two basic protocols: broadcasting and unique label assignment. These two protocols enable a complete mapping of the network and can serve as key building blocks in more advanced protocols. We develop distributed asynchronous protocols as well as derive lower bounds on their communication complexity, total bandwidth complexity, and node label complexity. The resulting lower bounds are sometimes surprisingly high, exhibiting the complexity of topology extraction in directed anonymous networks.

**Keywords:** Anonymous networks, directed networks, distributed protocols.

## 1 Introduction

In this work we study the fundamental problems of broadcasting and mapping (label assignment and topology extraction) in directed anonymous networks. In such a network $G$, processors do not have unique identifiers, they execute identical protocols, and they have no knowledge of the topology of the network (even the size or bounds on it are unknown). The only knowledge available to a vertex is its own degree.

Anonymous and unknown networks have been extensively studied during the last few decades. These studies include graph exploration [3, 1, 7], where a robot has to construct a complete map of an unknown environment; the study of network communication or more specifically the task of broadcasting and label assignment [4, 2, 5]; and various characterizations of attainable and unattainable tasks with respect to additional symmetry breaking assumptions [6, 8].

All studies above consider the underlying network to be undirected or directed but strongly connected. In such networks, the typical paradigm used in the context of anonymous networks is adaptive message passing. Namely, vertices in the network learn of each other via information transmitted up an down paths

[*]This work was supported in part by the Caltech Lee Center for Advanced Networking and by NSF grant ANI-0322475.

[†]Computer Science Division, The Open University of Israel, Raanana 43107, Israel. Email: `mikel@openu.ac.il`

[‡]Department of Electrical and Computer Engineering, Ben-Gurion University, Beer Sheva 84105, Israel. Email: `schwartz@ee.bgu.ac.il`

[§]California Institute of Technology, 1200 E California Blvd., Pasadena, CA 91125, U.S.A. Email: `bruck@paradise.caltech.edu` that connect these vertices, and construct their outgoing messages based on the information gained so far. In this work we address the design of distributed protocols when the underlying network is directed but not necessarily strongly connected. This setting differs substantially from the undirected case as a vertex may only have a single chance to transmit an outgoing message, and thus the protocols must be designed accordingly.

## 1.1 Our contribution

This work initiates the study of distributed asynchronous protocols over directed anonymous networks that are not necessarily strongly connected. To this end, we assume that the network $G$ has two *special* vertices, a root $s$ and a terminal $t$; and we assume that there is a path connecting $s$ and $t$. Our objective is to preform certain tasks on $G$. Roughly speaking, the protocols we present proceed as follows. An initial message is sent from the root vertex $s$ to its children. This initiates the distributed protocol which ends when the terminal vertex is in a final state with its state as output.

The algorithmic tasks we present in this work are the basic tasks of broadcasting and label assignment. For broadcasting, a message $m$ is given at the root $s$, and we wish to distribute $m$ throughout the entire network $G$. This, in itself, seems a trivial task obtained by simple propagation of $m$. However, we also want the protocol to terminate iff all the vertices of $G$ have received $m$. This turns out to be significantly more involved as our protocol cannot use the standard termination techniques used for graphs which are strongly connected (namely, *message passing*). We start by presenting a simple broadcasting protocol that will terminate correctly if $G$ is essentially a tree. We then present the main result of this work, a distributed asynchronous broadcasting protocol for general directed graphs. Finally we turn to design a protocol which assigns unique labels to all vertices of $G$ (thus enabling us to map the graph topology) and terminates after these labels have been assigned. Our label-assigning protocol is strongly built on our broadcasting protocol, and can be viewed as a slight modification of it. All the protocols we present are distributed and asynchronous. We consider several measures of complexity (which will be discussed in greater detail later): total communication complexity – the number of bits transmitted throughout the protocol, required bandwidth – maximal number of bits transmitted over a single edge, and the maximal number of bits in a label. Our results can be roughly summarized as follows:

**Broadcasting in acyclic networks:** A graph $G$ is said to be a *grounded tree* if every vertex of $G$ has in-degree 1, excluding the source $s$ with no incoming edges and the terminal $t$ which may have several incoming edges.

For grounded trees $G$, we describe an asynchronous distributed protocol that broadcasts a message $m$ from the root vertex $s$ to all of $G$, and halts iff all vertices have received $m$. The protocol has required bandwidth $O(\log|E|) + |m|$ and total communication complexity at most $O(|E|\log|E|) + |E|\,|m|$ (here $|m|$ is the size of $m$). We show that our results are tight by providing a matching lower bound on the total communication complexity of broadcasting in grounded trees.

For general directed acyclic graphs we provide an asynchronous distributed protocol for broadcasting with required bandwidth $O(|E|) + |m|$ and therefore total communication complexity $O(|E|^2) + |E|\,|m|$. Our results are the best possible when considering certain protocols we refer to as commodity preserving.

**Broadcasting in general networks:** For general $G$, we describe an asynchronous distributed protocol that broadcasts a message $m$ from the root vertex $s$ to all of $G$, and halts iff all vertices have received $m$. The protocol has total communication complexity at most $O(|E|^2\,|V|\log d_{\text{out}}) + |E|\,|m|$ and $O(|E|\,|V|\log d_{\text{out}}) + |m|$ required bandwidth (here and in what follows, $d_{\text{out}}$ is the maximal out-degree in the given network).

**Unique label assignment in general networks:** For general $G$, we describe an asynchronous distributed protocol that assigns unique labels to all vertices of $G$, and halts iff all vertices have been assigned labels. The protocol has total communication complexity at most $O(|E|^2 |V| \log d_{\text{out}})$, and $O(|E| |V| \log d_{\text{out}})$ required bandwidth. The labels have length of $O(|V| \log d_{\text{out}})$ bits which is also shown to be tight by proving a lower bound.

The resulting label complexity in the last protocol is surprisingly high. One might expect to be able to label the vertices with at most $O(\log |V|)$ bits, which is also achievable in both the undirected case, and the directed but strongly-connected case. However, the lower bound we describe shows our result to be tight, and the exponential blowup necessary.

## 1.2 Proof Techniques

We describe the main ideas behind our broadcasting protocols, label assignment protocol and lower bounds.

**Broadcasting:** Our objective is two-fold. First of all we want to transmit a message $m$ from the root $s$ to all vertices of $G$. This is done by propagation, every vertex that receives $m$ forwards it on its outgoing edges. Secondly, we want the protocol to terminate iff all vertices have received $m$. To this end, each internal vertex that has received $m$ will send out an additional message or messages that eventually will reach the terminal vertex $t$. These additional massages should be constructed in such a way that $t$ will be able to decide whether all vertices of $G$ have been visited or not. This would be a trivial task if the vertices of $G$ were to have unique labels and the terminal $t$ would know the size of $G$ (or even a bound on it), namely each vertex could simply send its label which will eventually reach $t$. However, in our model of study we assume both that vertices in $G$ do not have unique labels, and that they do not have an estimate on the size of $G$.

We now turn to the task of termination. Clearly, only using propagation, the terminal $t$ has no idea when the entire graph has been visited. Thus, additional information should be sent from the internal vertices in $G$ to $t$. Roughly speaking, this additional information will represent a certain commodity. Furthermore, the actions taken by each vertex in $G$ are *commodity-preserving*. Namely, each internal vertex partitions the commodity it receives among its outgoing edges. Thus, if the source were to send a unit of commodity into the graph $G$ and over time the terminal were to receive a unit of commodity, one could conclude that all vertices in $G$ have been visited.

For example, first consider the case of grounded trees. Here the commodity we use is a scalar value, and the intuition behind our protocol is that of commodity-preserving network flows. Namely, the source starts by transmitting as termination information the value 1 (here we assume the source has a single outgoing edge). This corresponds to a flow of value 1 leaving the source. When a vertex of out-degree $d$ receives the value $x$ as termination information, it sends the value $x/d$ on each of its outgoing edges (corresponding to the distribution of the flow of value $x$ it received). The terminal $t$ will declare termination once the values it receives on its incoming edges sum up to 1. Such a protocol will indeed work, however it will not achieve the tight communication complexity stated earlier. In Section 3, we improve on this protocol by using a different rule that governs the flow distribution at internal nodes.

But what happens when $G$ is not a grounded tree? The main difficulty lies in the case in which $G$ contains cycles. In this case an internal vertex $v$ may receive multiple commodities over time, some of which need to be discarded as they have already been viewed by $v$. To deal with cyclic networks, we use the commodity-preserving paradigm as before, but our commodity is more involved. Instead of transmitting as termination information a scalar value, we transmit multiple values. Loosely speaking, the commodity leaving the source is the unit interval $[0, 1)$ (represented by its end points). Each vertex, after receiving an interval will send on its $d$ outgoing edges a $d$-partition of the interval received. The terminal will declare termination only after it has received (over time) a family of intervals whose union is exactly the interval $[0, 1)$. Each internal vertex can detect cycles by *remembering* which intervals it has seen so far. Once a cycle is detected, additional information is sent to the terminal which guarantees correct termination. A naive implementation of these ideas will lead to exponential (in $|V|$) communication complexity. To obtain the polynomial complexity stated above, our protocols use several additional ideas.

**Label assignment:** As mentioned earlier, our unique label assignment protocol is strongly based on the commodity preserving paradigm used in the broadcast protocol discussed above. However, roughly speaking, now once a vertex $v$ in the graph receives a certain commodity, it partitions this commodity not only among its outgoing edges but rather a portion of the commodity is also preserved for $v$ itself. The commodity preserved for $v$ will act as a distinct label for the vertex $v$. To ensure correct termination of the protocol (once part of the commodity is *stored* at internal vertices of the graph) some changes to the broadcast protocol are needed.

**Lower bounds:** We present both lower bounds on the total communication complexity of any broadcasting protocol and lower bounds on the length of labels in any label assignment protocol. For our lower bound on the communication complexity, we first bound the number of distinct messages that must be transmitted as termination information throughout the protocol. Roughly speaking, we show that if the termination information of a given protocol is limited to exist in a small message space (say $\Sigma$), then one can design a graph $G$ on which the protocol will either terminate before all vertices have received the broadcast or not terminate at all. A detailed analysis of this line of proof appears in Section 3.2. Once we have established a bound on $|\Sigma|$, a bound on the communication complexity essentially follows by noticing that (under any encoding) most of the distinct messages $\sigma \in \Sigma$ must be represented by $\simeq \log |\Sigma|$ bits.

For our lower bounds on the label size of any label assignment protocol we use a standard pruning technique (e.g., [4]). Namely, we start by considering an exponentially large graph for which any protocol must assign labels of polynomial size. We then prune the graph in such a way that for certain vertices the protocol behaves exactly as if it were running on the original large graph. The pruned graph will on one hand have vertices which are assigned *large* labels and on the other hand have few vertices. This will imply our result.

### 1.3 Organization

The paper is organized as follows. In Section 2 we present our model and working assumptions. We continue in Section 3 to present our broadcasting protocol for grounded trees and DAGs, along with the proof that our protocol is optimal. In Section 4, we present our broadcasting protocol for general graphs. Finally, our label-assigning protocol is presented in Section 5. We conclude with a discussion and open problems.

## 2 The Model

In this work we study anonymous protocols on directed graphs $G = (V, E)$ with two special vertices. The root vertex, denoted by $s$ and the terminal vertex denoted by $t$. Vertices in $V \setminus \{s, t\}$ will be referred to as internal vertices. We assume that $s$ has no incoming edges and only one outgoing edge, and $t$ has no outgoing edges. As our access to $G$ is only through $s$ and $t$, if there are internal vertices which are not on a path from $s$ to $t$ but are still reachable from $s$ or connected to $t$, our protocols will not terminate. Throughout, to simplify our presentation, we assume that all vertices in $G$ are reachable from $s$.

All vertices in $G$ are assumed to know nothing of the topology of the network (including its size) nor do they have unique identifiers. Each vertex is assumed to know how many incoming and outgoing edges it has, and has the power to distinguish between different incoming/outgoing edges. The model we present is asynchronous. Our results can be easily extended to the case in which there are multiple root/terminal vertices, the root has multiple outgoing edges, the case in which there are vertices in $G$ that are not reachable from $s$, and to the case that the communication throughout the network is synchronous.

**Anonymous protocols** An anonymous protocol on $G$ is defined by the following primitives: a state space $\Pi$, a message space $\Sigma$, an initial state $\pi_0 \in \Pi$, an initial massage $\sigma_0$, a state function $f : \Pi \times \Sigma \times \mathbb{N} \to \Pi$, a message function $g : \Pi \times \Sigma \times \mathbb{N} \times \mathbb{N} \to \{\Sigma, \phi\}$, and a stopping predicate $S : \Pi \to \{0, 1\}$.

An anonymous protocol is executed as follows. Initially we associate the state $\pi_0$ with every vertex in $G$. The message $\sigma_0$ is sent on the outgoing edge of $s$. Each vertex that receives message $\sigma$ on its $i$-th incoming edge while having current state $\pi$, moves to state $\pi' = f(\pi, \sigma, i)$ and sends message $g(\pi, \sigma, i, j)$ on its $j$-th outgoing edge. If in the scenario above $g(\pi, \sigma, i, j) = \phi$, then no message is sent on outgoing edge $j$. We say that an anonymous protocol has terminated if $S(\pi) = 1$ for the state $\pi$ of $t$, in this case $\pi$ is the output of the protocol.

**Quality** There are several quality parameters that may be considered when studying anonymous protocols. The size of the state space is related to the amount of memory needed at each vertex of the network. The size of the message space is a bound on the maximum message length transmitted on edges on the network (i.e., bandwidth). Multiplying the bandwidth by the total number of messages sent over the network before termination will imply an upper bound on the total communication complexity of the protocol. In a synchronous model one may also consider the time it takes for the protocol to terminate.

In this work we focus on the asynchronous model in which we seek to design anonymous protocols with minimal total communication complexity. To this end, we study both the total number of messages transmitted throughout the network, and the maximum message size. There is an obvious trade-off between the two, and their product is the total communication complexity.

## 3 Broadcasting over Grounded Trees and DAGs

### 3.1 Broadcasting on grounded trees

We start with the task of broadcasting a message $m$ over grounded trees. The protocol will terminate iff all vertices in the network have received the broadcast message $m$. Roughly speaking, the protocol starts when the root vertex sends the message $m$ and additional *termination* information on its outgoing edge. Each vertex in the graph, will transmit information on its outgoing edges only when it has received the message $m$ from its incoming edge. A vertex of out-degree $d$ will transmit both the message $m$ and termination information which depends on the incoming termination information and the value $d$.

As stated in the Introduction, the intuition behind the protocol is that of commodity-preserving network flows. Namely, the source starts by transmitting as termination information the value 1. This corresponds to a flow of value 1 leaving the source. When a vertex of out-degree $d$ receives the value $x$ as termination information, it sends the value $x/d$ on each of its outgoing edges (corresponding to the distribution of the flow of value $x$ it received).

A naive implementation of this protocol results in total communication complexity bounded by $O(|E|^{3/2}) + |E|\,|m|$. The $|E|\,|m|$ factor corresponds to the message $m$ and is inevitable. The remaining $O(|E|^{3/2})$ follows from the fact that we must be able to represent many different $x$ values sent as termination information. Using a slightly different rule that governs the flow distribution at internal nodes, we are able to reduce this complexity to $O(|E| \log |E|))$ (which we show to be optimal). In the rule we present, the values $x$ transmitted as termination information will always be a power of 2. This way, $x$ can be represented efficiently.

More specifically, the flow distribution of our protocol proceeds as follows. When a vertex of out-degree $d$ receives the value $x$ as termination information, it sends the value $\frac{x}{2^{\lceil \log d \rceil}}$ on its first $2d - 2^{\lceil \log d \rceil}$ outgoing edges and the value $\frac{x}{2^{\lceil \log d \rceil - 1}}$ on its remaining outgoing edges. It is not hard to verify that such a distribution rule is commodity preserving and that the $x$ values transmitted will always be a power of 2. Indeed, for

$d \in \mathbb{N}$ and $\alpha = 2d - 2^{\lceil \log d \rceil}$ it holds that $\alpha \cdot 2^{-\lceil \log d \rceil} + (d - \alpha) \cdot 2^{-\lceil \log d \rceil + 1} = 1$. The terminal $t$ will declare termination iff the sum of values $x$ it receives on its incoming edges equals 1.

**Theorem 3.1** *The protocol above terminates iff each vertex of $G$ is connected to $t$. On termination each vertex in $G$ will have received the message $m$. The total communication complexity of the protocol is bounded by $O(|E| \log |E|) + |E|\,|m|$ (here $|m|$ is the size of $m$).*

**Proof.** Let $G$ be a grounded tree in which each vertex in $G$ is connected to the terminal $t$. During the execution of our protocol, each edge $(u, v)$ will carry a message only once: immediately after $u$ receives the message $(m, x)$ on its sole incoming edge. If $u$ has out-degree $d$, the messages leaving $u$ are of the form $(m, \frac{x}{2^{\lceil \log d \rceil}})$ or $(m, \frac{x}{2^{\lceil \log d \rceil - 1}})$. The termination values leaving $u$ sum up to the incoming termination value of $x$. This corresponds to the *flow conservation* described earlier and implies that, eventually, after each vertex has been visited, the sum of all termination values entering the terminal will be equal to 1. If at some point in time there is a vertex that has not yet been visited, then for similar reasons the sum of all termination values that have entered the terminal so far will be strictly less than 1 and thus the terminal will not be in an accepting state.

In the case that there exist vertices in $G$ that are not connected to $t$ (but reachable from $s$), we are again guaranteed that the protocol will not terminate. Indeed, the termination value $x > 0$ entering such a vertex will not reach $t$, implying that the sum of termination values entering the terminal will not reach the sum of 1.

For the communication complexity we bound the size of $\Sigma$ (the message space), which consists of the message $m$ and the termination values $x \in [0, 1]$. However, in the design of our protocol, $x$ is always a power of 2. Moreover, when applied to graphs with $|E|$ edges, the size of $x$ is at least $2^{-O(|E|)}$. This implies the asserted complexity. □

## 3.2 Lower bounds for broadcasting on grounded trees

We now turn to show that our broadcasting protocol is optimal for grounded trees. The proofs of the claims that follow appear in Appendix A. The outline of our proof is given in Section 1.2 of the Introduction.

**Theorem 3.2** *Any broadcasting protocol that (1) terminates on grounded trees $G$ in which each vertex of $G$ is connected to $t$, and (2) does not terminate otherwise; must have total communication complexity at least $\Omega(|E| \log |E|) + |E|\,|m|$. Namely, for any broadcasting protocol as above, there exists an infinite family of graphs $G$ for which the communication complexity of the protocol when executed on $G$ is at least $c\,|E| \log |E| + |E|\,|m|$ for some universal constant $c > 0$.*

**Proof.** Consider any broadcasting protocol that satisfies the requirements specified in the statement of the theorem with alphabet $\Sigma$. For a given graph $G$, we denote by $\Sigma_G \subseteq \Sigma$ the set of symbols transmitted over the edges of $G$ when the protocol is applied to $G$. Let $\Sigma_n$ be the union of $\Sigma_G$ over graphs $G$ with $n$ edges. In what follows we present lower bounds on the size of $\Sigma_n$. Namely, we present for infinitely many values of $n$ a family of graphs with $|E| = n$ for which the union of $\Sigma_G$ over these graphs satisfies the lower bound. Once we have established a lower bound on $\Sigma_n$, a bound on the total communication complexity easily follows.

**Lemma 3.3** *In any grounded tree $G$ we may assume w.l.o.g. that during any broadcasting process, each vertex of $G$ transmits a single message.*

Lemma 3.3 implies that the decision to declare termination of the protocol is based solely on the multiset of symbols entering the terminal node $t$. Thus, for a protocol $A$ we define the *termination set* $T_A$ to be the set of all finite multisets over alphabet $\Sigma$ which are terminating, namely, if the terminal node were to see exactly those symbols on its incoming edges it would declare termination.

Let $\sigma_A(e) \in \Sigma$ denote the character transmitted on edge $e$ when the protocol $A$ is executed. Similarly, let $\sigma_A(E')$ denote the *multiset* of symbols transmitted over edges in $E' \subseteq E$. We say that vertex $v_1 \in V$ is an *ancestor* of vertex $v_2 \in V$ if there is a path from $v_1$ to $v_2$. In that case we also say $v_2$ is a *descendant* of $v_1$. The following definition will help us construct the main tool of the proof.

**Definition 3.4** *Let $G = (V, E)$ be a directed acyclic graph with source $s$ and terminal node $t$. A* linear cut *is a partition of $V$ into two disjoint non-empty sets $V_1$ and $V_2$ such that there are no $v_1 \in V_1$ and $v_2 \in V_2$ where $v_1$ is a descendant of $v_2$.*

Obviously, in any linear cut $s \in V_1$ and $t \in V_2$. Linear cuts are useful since they provide us with possible snapshots of the symbols crossing the cut in some running of the protocol. This is shown in the following lemma.

**Lemma 3.5** *Let $E'$ be the set of edges crossing some linear cut of $G$. Then $\sigma_A(E') \in T_A$, i.e., any multiset of symbols crossing a linear cut when running protocol $A$, is terminating.*

Having seen the connection between terminating sets and linear cuts, we can state in the following theorem which is the main tool used in our proof.

**Theorem 3.6** *Let us consider two distinct linear cuts (not necessarily even in the same grounded tree!) with sets of edges crossing them $E'$ and $E''$ respectively. Then for any protocol, $\sigma_A(E') \not\subset \sigma_A(E'')$ (where the strict-subset notation $\subset$ is taken in the multiset sense).*

Lemma 3.5 and the resulting Theorem 3.6 apply equally well (though with some extra work) to directed acyclic graphs, and are the main tools we use for proving lower bounds. A simple application of Theorem 3.6 is given in the following lemma.

**Lemma 3.7** *Let $G = (V, E)$ be a grounded tree. If $e' \in E$ is an ancestor of $e'' \in E$, and at least one of the vertices on the path between them has out-degree at least $2$, then $\sigma_A(e') \neq \sigma_A(e'')$.*

We are now at a position to complete the proof of Theorem 3.2. For any $n \in \mathbb{N}$ we construct a grounded tree $G_n = (V_n, E_n)$ with $V_n = \{s, t, v_1, \ldots, v_n\}$ and edges

$$E_n = \{(s, v_1)\} \cup \{(v_i, v_{i+1}) \mid 1 \leqslant i \leqslant n-1\} \cup \{(v_i, t) \mid 1 \leqslant i \leqslant n\}.$$

Thus, $G_n$ has $n+2$ vertices and $2n$ edges (see Figure 5 of Appendix C).

According to Lemma 3.7, any protocol running on $G_n$ requires an alphabet of size at least $n+1$. It follows that $|\Sigma_n| = \Omega(n)$. Now, no matter how we encode the elements of $\Sigma$, it must be the case that at least $\Omega(n)$ elements will require a representation of at least $\Omega(\log n)$ bits. Thus, as $|E| = \Omega(n)$, the total required bandwidth and communication complexity of any protocol is $\Omega(\log |E|) + |m|$ and $\Omega(|E| \log |E|) + |E|\,|m|$ respectively (the additional dependence on $|m|$ follows from the need to broadcast the message $m$). $\square$

## 3.3 Broadcasting on directed acyclic graphs

As we move on to directed acyclic graphs, the overall picture is not so well understood. On the positive side, a straightforward modification of the commodity-preserving protocol for grounded trees presented earlier (in which the commodity is a scalar value) implies a broadcasting protocol for DAGs with required bandwidth $O(|E|) + |m|$ and therefore total communication complexity $O(|E|^2) + |E|\,|m|$. For a lower bound: we can assume w.l.o.g. that each node does not send out any messages until hearing a message on each of its incoming edges. (We omit the rigorous proof of both claims above.) Under this assumption, the linear-cut method presented for grounded trees applies, and we get a lower bound on the required bandwidth from each edge, of $\Omega(\log |E|)$ bits. Unfortunately, closing this gap currently seems out of reach. However, as we will see, if we restrict ourselves to commodity-preserving protocols our results are tight (namely we obtain a lower bound that matches the upper bound stated above). We suspect that an analogous lower bound can be proved for general protocols as well. The proof of the following theorem appears in Appendix B.

**Theorem 3.8** *Any commodity-preserving protocol requires bandwidth of at least $\Omega(|E|)$ bits.*

## 4 Broadcasting over General Graphs

We now turn to address the case of broadcasting over general graphs. Loosely speaking, the protocols we present generalize the commodity-preserving flow of the previous section, only the commodities are now distinct and uniquely identifiable. This basic commodity is given in the following definition.

**Definition 4.1** *The* interval set *over $[0,1)$ is defined as $\mathcal{I}[0,1) = \{[a,b) \subseteq [0,1) \mid a,b \in \mathbb{R}\}$. Similarly, the* interval-union *over $[0,1)$, denoted as $\mathcal{U}[0,1)$, is defined as the set of all finite unions of disjoint intervals from $\mathcal{I}[0,1)$. Elements of $\mathcal{U}[0,1)$ will be called* interval-unions. *A* partition *of an interval-union $\alpha \in \mathcal{U}[0,1)$ is a finite set of disjoint interval-unions whose union is $\alpha$.*

For our purposes, an interval $[a,b)$ may be simply represented by the numbers $a$ and $b$, and since we will have a choice in selecting $a$ and $b$, we will choose them to be binary-point numbers of finite representation, i.e., a sum of powers of 2 with a finite number of summands. By convention, an interval of the form $[a,a)$ is the unique empty interval which we consider a subset of any interval.

We are now in a position to state our broadcasting protocol. Loosely speaking, both the messages transmitted throughout the protocol and the state space consist of elements $\alpha \in \mathcal{U}[0,1)$. The protocol starts when the root vertex $s$ transmits as its initial message the interval-union consisting of the single interval $[0,1)$. Each vertex has a state $\alpha$ which includes the union of all the intervals it has *seen* so far. When a vertex receives as a message an interval-union $\alpha'$, it adds $\alpha'$ to its state $\alpha$ and then partitions its new state among its outgoing edges. In such a way the interval $[0,1)$ is partitioned among the vertices of the graph (vertices may have corresponding $\alpha$'s that intersect). Our protocol will terminate once $t$ has seen the entire interval $[0,1)$. However, if there exists a cycle in the graph, part of the interval may get "stuck" in a loop never reaching the terminal $t$. Our protocol identifies such cases and through simple propagation "notifies" $t$ to disregard this missing part. A crucial aspect of our protocol is the so-called *state-monotonicity* property, which ensures that for every vertex $v$, the interval-union which is the state of $v$ at time $T$ is included in the interval-union which is the state of $v$ at any time $T' > T$.

More specifically, our state space $\Pi$ consists of the set $\{\mathcal{U}[0,1)\}^*$. Namely, each vertex $v$ of out-degree $d$ holds a state $\pi = (\bar{\alpha}, \beta)$, where $\bar{\alpha} = (\alpha_1, \alpha_2, \ldots, \alpha_d) \in \{\mathcal{U}[0,1)\}^d$ and $\beta \in \mathcal{U}[0,1)$. Intuitively, each $\alpha_j$ corresponds to the interval-union previously sent on the $j$-th outgoing edge of $v$; and $\beta$ corresponds to additional information needed to cope with cycles. The message space will consist of two union-intervals, namely $\Sigma = (\alpha, \beta) \in \mathcal{U}[0,1) \times \mathcal{U}[0,1)$, and a message $m$ (to be broadcast). In our protocol, each message sent includes the message $m$. In what follows, we focus on the additional information in $\mathcal{U}[0,1) \times \mathcal{U}[0,1)$ sent in a message $\sigma$, and for simplicity of notation, we do not mention the message $m$. The initial message sent by $s$ is $\sigma_0 = ([0,1), [0,0))$. We assume that the initial state of all vertices is $\pi_0 \in (\{[0,0)\}^*, [0,0))$. Again, each vertex $v$ of out-degree $d$ holds initial state $(\alpha_0, \beta_0) = ([0,0)^d, [0,0))$.

Now we come to define the functions $f$ and $g$. Consider a vertex $v$ of out-degree $d$. Let $\pi = (\bar{\alpha}, \beta) = ((\alpha_i)_{i=1}^d, \beta)$ be the state of $v$, and let $\sigma = (\alpha', \beta')$ be the message $v$ receives on incoming edge $i$. We define the value $\pi'' = (\bar{\alpha}'', \beta'') = ((\alpha''_j)_{j=1}^d, \beta'')$ of $f(\pi, \sigma, i)$. We start by defining a canonical partition of the union-interval $\alpha'$ into $d$ parts. Let $\alpha' = \bigcup_{k=1}^r I_k$ where each $I_k$ is a non-empty interval of the form $[a,b)$. Let

$I_1^1, \ldots, I_1^{d-1}$ be a partition of the first interval $I_1$ into $d-1$ parts. A canonical partition of $\alpha'$ into $d$ union intervals $\alpha_1^*, \ldots, \alpha_d^*$ is defined by $\alpha_j^* = I_1^j$ for $j \in \{1, \ldots, d-1\}$ and $\alpha_d^* = \bigcup_{k=2}^r I_k$.

Now, if $\pi = \pi_0$ then $\bar{\alpha}''$ is a canonical partition of $\alpha'$ into $d$ parts, and $\beta'' = \beta'$. If $\pi \neq \pi_0$ then for $j \in \{1, \ldots, d-1\}$ the interval-union $\alpha_j'' = \alpha_j$ (i.e., $\alpha_j''$ remains unchanged), and the interval-union $\alpha_d'' = \alpha' \cup \alpha_d \setminus \bigcup_{j=1}^{d-1} \alpha_j$. The interval-union $\beta'' = \beta' \cup \beta \cup \bigcup_{j=1}^{d} (\alpha' \cap \alpha_j)$. The function $g$ is defined entirely by $\pi''$, namely $g(\pi, \sigma, i, j) = g((\bar{\alpha}, \beta), (\alpha', \beta'), i, j) = (\alpha_j'' \setminus \alpha_j, \beta'' \setminus \beta)$.

We elaborate on the definition of $f$ and $g$. We start by noticing that the state $\pi = (\bar{\alpha}, \beta)$ of each vertex is increasing (with respect to set inclusion) over time. Both in every $\alpha_j$ portion and in the $\beta$ portion of $(\bar{\alpha}, \beta)$. For $j \in \{1, \ldots, d-1\}$, $\alpha_j$ is changed only once in the entire running process, while $\alpha_d$ may change many times. $\beta$ may also change many times during the running of the protocol. It is important to notice that a message is sent from $v$ on outgoing edge $j$ iff either $\alpha_j$ changes or $\beta$ changes.

Finally, the predicate $S(\pi) = 1$ iff $\pi = (\alpha, \beta)$ and $\alpha \cup \beta = [0, 1)$.

**Theorem 4.2** *The protocol above terminates iff each vertex of $G$ is connected to $t$. On termination each vertex in $G$ will have received the message $m$. The total communication complexity of the protocol is $O(|E|^2 \, |V| \log d_{\text{out}}) + |E| \, |m|$.*

**Proof.** Let $(\bar{\alpha}, \beta)$ be the state of a vertex $v$ in $G$ at some time $T$. Let $a \in \mathbb{R}$. We say that the $j$-th outgoing edge of $v$ $\alpha$-carries $a$ at time $T$ if $a \in \alpha_j$ at that time. We say that the $j$-th outgoing edge of $v$ $\beta$-carries $a$ at time $T$ if $a \in \beta$ at that time. Notice that if an edge $\alpha$-carries ($\beta$-carries) a value $a$ at time $T$, it continues carrying it at time $T' > T$ due to our state-monotonicity.

For $a \in [0, 1)$, consider the edges $\alpha$-carrying $a$ over time $T$. These edges form a (monotonically increasing over time) subgraph $G_T(a)$ of $G$ in which the root has out-degree 1, vertices in $G$ with in-degree zero have out-degree zero, and the remaining vertices have out-degree at most 1, i.e., $G_T(a)$ is either a path between $s$ and some vertex $v$, or a path starting at $s$ which at some point closes a loop. This follows from the fact that for a state $\pi = ((\alpha_j), \beta)$ of $v$, the interval-unions $\alpha_j$ have an empty intersection, and their union is equal to the union of all union-intervals $\alpha'$ that have passed through $v$. Thus for each $a \in [0, 1)$ either there exists a time $T$ for which the subgraph $G_T(a)$ is connected to $t$, in which case $t$ will eventually receive $a$, or there exists a $T$ for which $G_T(a)$ contains a cycle which prevents $a$ form reaching $t$.

If for some $a$ and $T$ the corresponding subgraph $G_T(a)$ contains a cycle, then it is not hard to verify that by our definitions $a$ will be $\beta$-carried throughout the predecessors of the cycle. Specifically, $a$ will eventually be $\beta$-carried on edges leading to $t$. We conclude that $t$ will eventually be in a state $\pi$ for which $S(\pi) = 1$ (namely, the protocol will always eventually halt).

Assume that the protocol terminated at time $T$, while there was still a vertex $v$ in $G$ that has not been visited. Let $P$ be a path from $s$ to $v$ and let $u$ be vertex on this path closest to $s$ that has not been visited. By our definitions there is a value $a$ which is $\alpha$-carried on the path $P$ up to $u$ (including) but not $\alpha$-carried on any other edge along the path $P$ between $u$ and $v$. Namely, at time $T$ the graph $G_T(a)$ consists of a path that ends at the vertex $u$. Thus, $a$ has not been $\alpha$-carried to $t$ yet at time $T$. Theoretically, it may still be the case that $a$ has been $\beta$-carried to $t$, however in order for $a$ to be $\beta$-carried on any edge $e$ at time $T$, the graph $G_T(a)$ must contain a cycle, and thus no vertices with in-degree greater than 1 and out-degree 0, which contradicts the properties of the vertex $u$.

It is left to analyze the communication complexity of our protocol. This is done by analyzing both the total number of messages sent along the edges of $G$, and by analyzing the bit complexity of each message. We start with the latter.

**Theorem 4.3** *At most $O(|E| \, |V| \log d_{\text{out}}) + |m|$ bits are required to represent a symbol from the edge alphabet.*

**Proof.** As described previously, an interval $[a, b) \subseteq [0, 1)$ will be encoded by writing down the numbers $a$ and $b$. We will always make sure that $a$ and $b$ are binary-point numbers of finite representation. Given an interval $[a, b) \subseteq [0, 1)$, say we want to partition it into $k \geqslant 1$ disjoint intervals. Let $N$ be the smallest power of 2 such that $N \geqslant k$ and denote $\Delta = (b - a)/N$. We partition $[a, b)$ in the following manner:

$$[a, a + \Delta), [a + \Delta, a + 2\Delta), \ldots, [a + (k-2)\Delta, a + (k-1)\Delta), [a + (k-1)\Delta, b)$$

i.e., into $k - 1$ intervals of size $\Delta$ and one interval of size $(b - a) - (k - 1)\Delta$. Each of the new intervals requires additional $O(\log k)$ bits to encode relative to the encoding of $[a, b)$.

Consider any interval-union transmitted on an edge in $G$. Let $d_{\text{out}}$ be the maximal out-degree of any vertex in $G$. We show that (a) the number of intervals in this union is bounded by $|E|$, and (b) the endpoints of any interval in this interval-union can be represented by $O(|V| \log d_{\text{out}})$ bits. This suffices to prove our assertion.

Let $a$ and $b$ be the end points of an interval in an interval-union transmitted in $G$. The bit representation of $a$ and $b$ is an artifact of how may times the intervals that included $a$ and $b$ were partitioned during the execution of our protocol. As each vertex in $G$ only preforms interval partitioning once during the execution of the protocol (only when it received a message for the first time), and this partition is into at most $d_{\text{out}}$ parts, we conclude that the total bit representation of $a$ and $b$ is $O(|V| \log d_{\text{out}})$.

We now bound the number of intervals which constitute an interval-union. In general, this is done by bounding the total number of different intervals viewed in our protocol. A new interval may be formed only by canonical partitioning of the $\alpha$ part, or by the intersection of two intervals in the $\beta$ part. At most $O(|E|)$ intervals may be obtained by canonical partitioning (each vertex $v$ does the partitioning once, and produces $d_{\text{out}}(v)$ new intervals). By observing that the set of interval end-points in the $\beta$ part must be a subset of the interval end-points of the $\alpha$ part, we conclude that the $\beta$ part is also made up of at most $O(|E|)$ intervals. $\square$

Continuing the proof of Theorem 4.2, an essential observation is the fact that for any edge, any $a \in \mathbb{R}$ is $\alpha$-carried ($\beta$-carried) by it at most once. It follows from the previous claim that the total communication *on any one edge* is $O(|E| |V| \log d_{\text{out}}) + |m|$, which concludes the proof. $\square$

## 5 Label Assignment on General Graphs

The protocol from the previous section, though polynomial in the parameters of the graph, seems over-complicated for the task achieved — broadcasting over general graphs. However, this lays the foundation for a protocol which assigns unique labels to vertices of $G$.

A slight variation to our previous protocol for broadcasting over general graphs allows us to assign unique id's to the vertices of $G$. In the label-assignment protocol, the state of each vertex will include an additional union-interval which represents its label. Namely, the state of each vertex of out-degree $d$ will now be $\pi = (\bar{\alpha}, \beta) = ((\alpha_j)_{j=0}^{d}, \beta)$ instead of $\pi = ((\alpha_j)_{j=1}^{d}, \beta)$ as before. The additional union-interval $\alpha_0$ will be the label of the vertex $v$. We need to define the functions $f$ and $g$ accordingly. $f(\pi, \sigma, i) = \pi'' = (\bar{\alpha}'', \beta'') = ((\alpha''_j)_{j=0}^{d}, \beta'')$ is essentially defined as before, but now if $\pi = \pi_0$ then $\bar{\alpha}''$ is a canonical partition of $\alpha'$ into $d + 1$ parts $(\alpha''_j)_{j=0}^{d}$, and $\beta'' = \beta' \cup \alpha_0$. If $\pi \neq \pi_0$ then $f$ is exactly as defined previously with the addition that the value of $\alpha''_0$ is identical to that of $\alpha'_0$. The function $g$ is defined as before.

**Theorem 5.1** *The unique labeling protocol above terminates iff each vertex of $G$ is connected to $t$. On termination each vertex in $G$ will have a unique label. The total communication complexity of the protocol is $O(|E|^2 |V| \log d_{\text{out}})$. Each resulting vertex label is of length $O(|V| \log d_{\text{out}})$ bits.*

**Proof.** The proof is essentially the same as that of Theorem 4.2 so we only give the intuition behind it. What this protocol does is send the interval $[0, 1)$ into the graph. Vertices receive interval-unions on incoming edges and send subsets of those on outgoing edges. But unlike the protocol described in Section 4, they do not partition the incoming interval-unions among the outgoing edges, but rather partition them among the outgoing edges and themselves, keeping a sub-interval to be used as a unique identification.

Thus, like the proof of Theorem 4.2, we follow the path which $\alpha$-carries some value $a \in [0, 1)$. Like before, this path may either end in an output vertex, or close up on itself because of a cycle, and then $a$ is $\beta$-carried to the output through simple propagation. Only now we have a third option, where $a$ stops being $\alpha$-carried not because of a cycle or because of reaching an output vertex, but rather because a vertex has taken an interval containing $a$ as its label. From then on, $a$ is being $\beta$-carried by propagation to the output. The monotonicity and the partitioning done by each vertex ensure that the resulting identification intervals are disjoint.

For the bit complexity of a label, notice that each label is a single interval, and use the analysis of Theorem 4.2. □

One might think that a labeling of $|V|$ vertices by labels of $O(|V| \log d_{\text{out}})$ is inefficient, but the following theorem shows that this number is actually required by any labeling protocol. The outline of our proof is given in Section 1.2 of the Introduction.

**Theorem 5.2** *Any labeling protocol giving unique labels to the vertices produces labels each of length $\Omega(|V| \log d_{\text{out}})$ bits.*

**Proof.** Let us consider a full tree of height $h$ and degree $d$ where the edges are directed away from the root $s$. Let $t$ be a vertex connected to all leaves of the tree. Any labeling protocol that gives unique labels to the vertices will use at least $d^h$ distinct labels. Thus, there exists some leaf vertex $v$, which receives a label of $\Omega(h \log d)$ bits.

The main observation is that in such a graph without cycles, each vertex receives a label which depends only on the messages sent to it along a path from the root. Therefore, we can remove all vertices of the original tree not on the path from the root to $v$, and connecting all other $d-1$ edges from each of the vertices on the path to the terminal $t$ (see Figure 6 of Appendix C).

Thus we get a new graph with a total of $h+3$ vertices and maximal out-degree $d$. By our previous observation, $v$ still receives a label of $\Omega(h \log d) = \Omega(|V| \log d_{\text{out}})$ bits. We stress that the execution of the labeling protocol on the path to $v$ in the new *pruned* graph is identical to the execution of the labeling protocol on the path to $v$ in the original graph. Indeed, let $u$ be a vertex along the path to $v$. The label of $u$ and the messages leaving $u$ depend solely on the incoming messages to $u$ and the number of outgoing edges of $u$: both remain unchanged. □

## 6 Conclusion and Open Problems

In this work we explored asynchronous distributed protocols running on anonymous directed networks. By showing how to broadcast and assign labels on such networks, we can transform anonymous networks to labeled networks and even map the whole topology by flooding local information available to nodes. This is a crucial step since most existing protocols require either some knowledge of the network or unique labeling of the nodes.

By comparing the results of this work with those over *undirected* anonymous networks, we can see a wide gap in performance which may be attributed mainly to the problem of termination, and the possible lack of feedback due to the directionality of edges. This is strongly pronounced in the case of assigning labels, where instead of $O(\log |V|)$ bits for labels in undirected anonymous networks (or even directed but strongly-connected networks) we require at least $\Omega(|V| \log d_{\text{out}})$, an exponential gap. This is directly attributed, as seen from the lower-bound proof, to the lack of feedback from terminal to source.

Another possible gap can be seen in the broadcasting protocol when moving from the restricted case of grounded trees, to the slightly more general case of directed acyclic graphs. Protocols which are commodity preserving suffer from an exponential increase in required bandwidth over the edges, from $O(\log |E|)$ to $\Omega(|E|)$ bits per edge. However, we do not know how to improve the lower bound for DAGs, and so, we pose as an open question: what is the required bandwidth per edge for broadcasting over DAGs? and over general graphs?

## References

[1] S. Albers and M. R. Henzinger. Exploring unknown environments. In *STOC '97: Proceedings of the twenty-ninth annual ACM symposium on Theory of computing*, pages 416–425, New York, NY, USA, 1997. ACM Press.

[2] B. Awerbuch, O. Goldreich, D. Peleg, and R. Vainish. A trade-off between information and communication in broadcast protocols. *Journal of the ACM*, 37(2):238–256, 1990.

[3] X. Deng and C. H. Papadimitriou. Exploring an unknown graph. *J. of Graph Th.*, 32:265–297, 1999.

[4] P. Fraigniaud, A. Pelc, D. Peleg, and S. Pérennes. Assigning labels in unknown anonymous networks. *Distributed Computing*, 14:163–183, 2001.

[5] L. Gargano, A. Pelc, S. Pérennes, and U. Vaccaro. Efficient communication in unknown networks. *Networks*, 38(1):39–49, 2001.

[6] T. Kameda and M. Yamashita. Computing on anonymous networks: Part I - characterizing the solvable cases. *IEEE Transactions on parallel and distributed systems*, 7(1):69–89, 1996.

[7] P. Panaite and A. Pelc. Exploring unknown undirected graphs. *J. of Algorithms*, 33:281–295, 1999.

[8] N. Sakamoto. Comparison of initial conditions for distributed algorithms on anonymous networks. In *PODC: 18th ACM SIGACT-SIGOPS Symposium on Principles of Distributed Computing*, pages 173–179, 1999.

# A Proof of Theorem 3.2

## A.1 Proof of Lemma 3.3

**Proof.** The proof is by a simple induction on the distance from the source $s$. The induction basis is obvious: the source sends only one message on each of its outgoing edges. For the induction step, it is easy to see that inner vertices (i.e., not the terminal vertex $t$), have only one incoming edges which, by the induction hypothesis, carries just one message. Thus, inner vertices send exactly one message on each of their outgoing edges. □

## A.2 Proof of Lemma 3.5

**Proof.** Let $G$ be a grounded tree, and $E'$ a set of edges crossing some linear cut which partitions $V$ into $V_1 \cup V_2$. By the definition of a linear cut, and due to the fact that we use the asynchronous model, we can easily think of $V_1$ as the set of vertices which already processed the symbols on their incoming edges (and sending symbols on their outgoing edges), and $V_2$ as the set of vertices which did not yet process incoming symbols. Thus, the linear cut resembles a snapshot in time of a possible running of the protocol.

At this point we can construct a different grounded tree $G^* = (V^*, E^*)$ with $V^* = V_1 \cup \{t\}$, all the edges between vertices of $V_1$ remain the same, and the edges crossing the linear cut in $G$, all connect to the terminal node $t$ (see Figure 1).

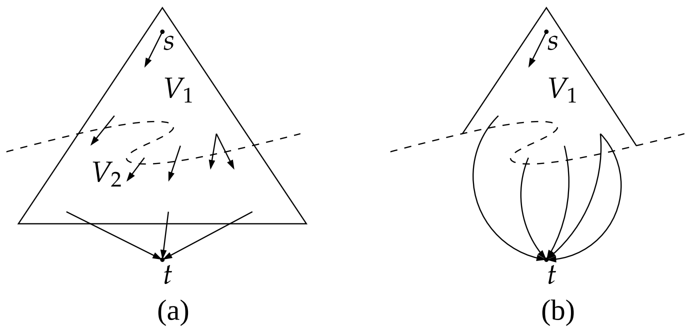

Figure 1: (a) The grounded tree $G$ with the linear cut partitioning it into $V_1$ and $V_2$. (b) The grounded tree $G^*$

Obviously, $G^*$ is also a grounded tree, and furthermore, all inner vertices are on a path from $s$ to $t$. It follows that the protocol should run on it correctly as well, and terminate. The sequence of running the vertices on $G$ until reaching the linear cut could be repeated when running on $G^*$ and so the multiset of symbols on edges crossing the cut in $G$ is exactly the same as that in $G^*$. Thus, the protocol should terminate upon getting $\sigma_A(E')$ when running on $G^*$, which by definition means that $\sigma_A(E') \in T_A$, as we wanted to prove. □

## A.3 Proof of Theorem 3.6

**Proof.** Assume to the contrary that $\sigma_A(E') \subset \sigma_A(E'')$. Let $G = (V, E)$ be the grounded tree from which $E''$ was obtained, and let $V_1 \cup V_2$ be the appropriate linear cut. In a similar fashion to the proof of Lemma 3.5, we construct a new grounded tree $G^* = (V^*, E^*)$, with $V^* = V_1 \cup \{t, t^*\}$ where $t$ is the terminal vertex and $t^*$ is an auxiliary vertex. All the edges between vertices of $V_1$ remain the same. Now, there exists a partition of $E''$ into two disjoint non-empty sets $E''_1 \cup E''_2$ such that $\sigma_A(E''_1) = \sigma_A(E')$. We connect the edges $E''_1$ to the vertex $t$, and the vertices of $E''_2$ to $t^*$ (see Figure 2).

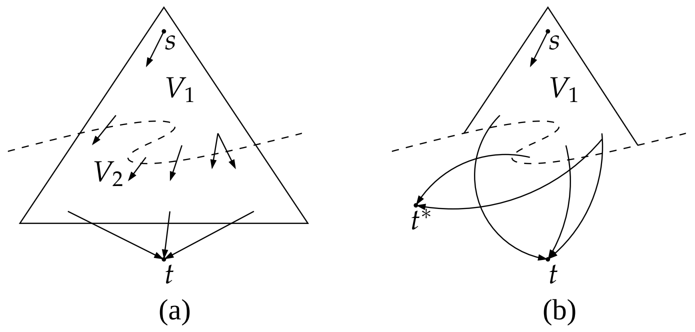

Figure 2: (a) The grounded tree $G$ with the linear cut partitioning it into $V_1$ and $V_2$. (b) The grounded tree $G^*$ and auxiliary node $t^*$

As before, running protocol $A$ on $G^*$ produces symbols identical to those produced when $A$ is run on $G$. By Lemma 3.5, $\sigma_A(E') \in T_A$, but we constructed $G^*$ so that $\sigma_A(E''_1) = \sigma_A(E')$, and so it follows that protocol $A$ will terminate when running on $G^*$. However, vertex $t^*$ is reachable from $s$ but is not connected to the terminal vertex $t$, and so a proper protocol should not terminate — a contradiction. Thus, $\sigma_A(E') \not\subset \sigma_A(E'')$. □

### A.4 Proof of Lemma 3.7

**Proof.** Let us choose arbitrarily a linear cut which the edge $e'$ crosses, and let the appropriate partition of $V$ be $V_1 \cup V_2$. Denote by $E'$ the edges crossing the cut. Since $e'$ is an ancestor of $e''$, there exists a path $v_0, v_1, \ldots, v_{k-1}, v_k$ such that $(v_0, v_1) = e'$ and $(v_{k-1}, v_k) = e''$. We note that by the definition of a linear cut, necessarily $v_0 \in V_1$ and $\{v_1, \ldots, v_k\} \subseteq V_2$.

We can now consider another partition of $V$ into $V_3 \cup V_4$ where,

$$V_3 = V_1 \cup \{v_1, v_2, \ldots, v_{k-1}\} \qquad\qquad V_4 = V_2 \setminus \{v_1, v_2, \ldots, v_{k-1}\}\,.$$

This partition is clearly another linear cut (see Figure 3). Denote by $E''$ the vertices crossing this cut. Now $e' \notin E''$ but $e'' \in E''$. Also, since we required at least one vertex from $\{v_1, \ldots, v_{k-1}\}$ to have an out-degree at least 2, it follows that $|E''| > |E'|$.

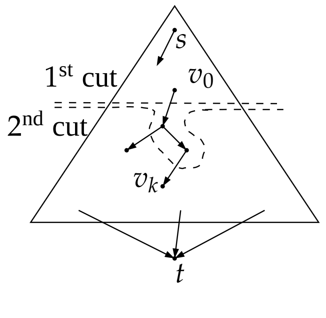

Figure 3: The two linear cuts of Lemma 3.7

Now, assume to the contrary that $\sigma_A(e') = \sigma_A(e'')$. In that case, necessarily, $\sigma_A(E') \subset \sigma_A(E'')$, which is impossible by Theorem 3.6. □

## B Proof of Theorem 3.8

As mentioned, we consider protocols in which each node does not send out any messages until hearing a message on each of its incoming edges. For such protocols we change the definition of the function $g$ that governs the outgoing messages. Namely, for a vertex of in degree $d_{\text{in}}$, $g$ is now defined as a function from $\Pi \times \Sigma^{d_{\text{in}}} \times \mathbb{N}$ to $\Sigma$. For incoming messages $\bar{\sigma} = \{\sigma_1, \ldots, \sigma_{d_{\text{in}}}\}$ and state $\pi$, $g(\pi, \bar{\sigma}, i)$ is the message transmitted on outgoing edge $i$ given incoming messages $\bar{\sigma}$ and state $\pi$.

**Definition B.1** *A protocol is commodity-preserving if there exists a real function $q : \Sigma \to \mathbb{R}^+$ such that for any in-degree $d_{\text{in}}$, any out-degree $d_{\text{out}}$, any state $\pi \in \Pi$, and any set of incoming messages $\bar{\sigma} = \{\sigma_1, \ldots, \sigma_{d_{\text{in}}}\}$*

$$\sum_{i=1}^{d_{\text{in}}} q(\sigma_i) = \sum_{i=1}^{d_{\text{out}}} q\left(g(\pi, \bar{\sigma}, i)\right).$$

**Proof.** Unlike previous proofs which considered a graph and manipulated it, we consider a set of graphs of some general form, and prove that at least one of those graphs causes the protocol to require a bandwidth of $\Omega(|E|)$ bits.

Let us fix a commodity-preserving protocol $A$. We build these graphs step by step, starting with the two vertices $s$ and $t$. We connect $s$ to $v_0$. Since we can easily scale the commodity, let us assume w.l.o.g., that the source $s$ sends $\sigma_0$ with $q(\sigma_0) = 1$ to $v_0$ under protocol $A$. We now draw two outgoing edges from $v_0$. Protocol $A$ must divide the commodity $\sigma_0$ between the two edges, and for convenience of illustration, let us assume the smaller quantity is transmitted over the left edge while the larger is sent over the right edge. Namely, if $v_0$ transmits $\sigma_1$ to $v_1$ and $\sigma_2$ to $u_0$ then $q(\sigma_1) \leqslant q(\sigma_2)$. We connect the left edge to a vertex we call $v_1$. We now continue in the same fashion: vertex $v_i$ has two outgoing edges and the protocol divides the quantity entering $v_i$ so that the smaller quantity flows over the left outgoing edge to $v_{i+1}$. We connect the right outgoing edge from $v_i$ to a vertex we call $u_i$. We continue this splitting until we reach $v_{2n-1}$ which we do not split anymore, but rather connect to $t$. We also connect all $u_i$ with $i \equiv 1 \pmod 2$ to $t$. The resulting graph is seen in Figure 4 (a).

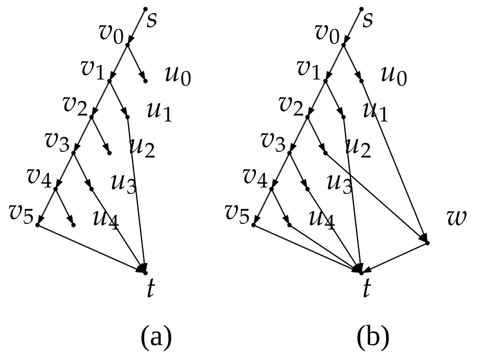

Figure 4: (a) The skeleton tree for $n = 3$ and (b) after choosing the subset $\{u_0, u_2\}$

We introduce an auxiliary vertex $w$ which we connect to $t$. We can now choose arbitrarily any subset $S \subseteq \{u_0, u_2, \ldots, u_{2n-2}\}$, and connect the vertices of $S$ to $w$, while the rest we connect to $t$ (see Figure 4 (b)). For each choice of $S$ we get a different graph. We note that if not for vertex $w$, the resulting graphs would all be grounded trees.

Let us denote by $q(v) = q(\sigma_v)$ the quantity flowing into vertex $v$ under Protocol $A$. We can write the following general inequality chain:

$$q(u_{2i+2}) < q(v_{2i+2}) \leqslant \frac{1}{2}q(v_{2i+1}) \leqslant \frac{1}{2}q(u_{2i}). \tag{1}$$

Since protocol $A$ is a commodity-preserving protocol, the quantity transmitted from $w$ to $t$ is simply $\sum_{v \in S} q(v)$. But by inequality (1), for two subsets $S, S' \subseteq \{u_0, u_2, \ldots, u_{2n-2}\}$, $S \neq S'$, it follows that

$$\sum_{v \in S} q(v) \neq \sum_{v \in S'} q(v)$$

and so we get as many different quantities flowing from $w$ to $t$ as there are subsets, i.e., $2^n$ different quantities implying $2^n$ different symbols $\sigma \in \Sigma$.

It follows that to encode $\Sigma$ we need $\Omega(n)$ bits. However, the graph itself contains $O(n)$ vertices and $O(n)$ edges. Hence the proof is complete. $\square$

## C Figures

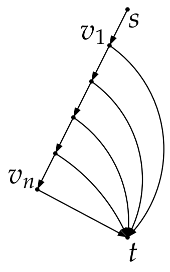

Figure 5: The grounded tree $G_n$

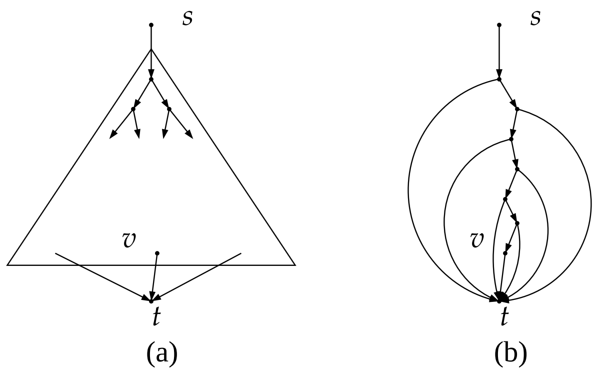

Figure 6: (a) The full tree and (b) the pruned tree, which are used in Theorem 5.2