MULTIPLE COMMUNICATION IN MULTI-HOP
RADIO NETWORKS

by

R. Bar-Yehuda, A. Israeli, A. Itai

Technical Report #537
February 1989
ABSTRACT

Two tasks of communication in a multi-hop synchronous radio network are considered: point-to-point communication and broadcast (sending a message to all nodes of a network). Efficient protocols for both problems are presented. Even though the protocols are probabilistic, it is shown how to acknowledge messages deterministically.

Let \( n, D, \) and \( \Delta \) be the number of nodes, the diameter and the maximum degree of our network, respectively. Both protocols require a setup phase in which a BFS tree is constructed. This phase takes \( O((n + D \log n) \log \Delta) \) time, after which \( k \) point-to-point transmissions require an average of \( O((k + D) \log \Delta) \) time. Therefore the network allows a new transmission every \( O(\log \Delta) \) time slots.

After the setup, \( k \) broadcasts require an average of \( O((k+D) \log \Delta \log n) \) time. Hence the average throughput of the network is a broadcast every \( O(\log \Delta \log n) \) time slots. Both protocols pipeline the messages along the BFS tree. They are always successful on the graph spanned by the BFS tree. Their probabilistic behavior refers only to the running time.

Using the above protocols the ranking problem is solved in \( O(n \log n \log \Delta) \) time. The performance analysis of both protocols constitutes a new application of queueing theory.
1. Introduction

A radio network is a network of processors which communicate using radio. The main characteristic of radio communication is that if a receiver is in the range of two or more transmitting stations then it does not receive anything. A radio communication network is single-hop if all nodes are in transmission range of each other. Otherwise it is multi-hop. Thus, sending a message between two stations in a multi-hop network might involve transmissions of intermediate stations.

Real life radio networks for data communication are quite limited. In fact, most such networks are single-hop. The only existing multi-hop networks resort to the tree topology. This situation looks rather odd considering the ease in which radio networks can be initiated and the flexibility and modularity of their operation.

A new approach for controlling the activity in multi-hop radio networks was presented in the work of [BG187], where an efficient broadcast protocol is presented. Their method gives a new way of looking at radio networks. However, they do not provide protocols for many important network tasks.

In the present work we use some of the ideas presented in [BG187], together with some new ideas to get very efficient protocols for two important and practical tasks. The tasks are $k$ point-to-point transmission and $k$-broadcast. Point-to-point transmission is the task of sending a message from one station to another. Broadcast is a task initiated by a single stations called the source, which transmits a message to all stations in the network. A $k$-point-to-point transmission ($k$-broadcast) is a task which consists of $k$ independent point-to-point transmissions ($k$ broadcasts). Besides the theoretical interest, these tasks constitute a major part of real-life multi-hop radio network.

1.1. Model Description  
Our model consists of an undirected graph whose nodes represent stations (i.e., processors) and whose edges indicate possible communication, (i.e., an edge between two nodes implies that the corresponding processors are within range of each other). The processor have distinct IDs. Initially, each processor knows its local neighborhood (i.e. the identity of its neighbors) the size of the network, $n$, and an upper bound $\Delta$ on the maximum degree of the network. It need not have any additional information of the topology of the network.

The processors may transmit and receive messages of length $O(\log n)$ and communicate in synchronous time-slots subject to the following rules. In each time-slot, each processor acts either as a transmitter or as a receiver. A processor acting as a receiver is said to receive a message in time-slot $t$ if exactly one of its neighbors transmits in time-slot $t$. The message received is the one sent. Since communication is synchronous the only difficulty in routing messages, in this model, is the possibility of conflicts; that is, situations when several neighbors of a processor transmit simultaneously and it receives nothing. More specifically, we assume that there is no conflict detection, (see [BG188]).
Throughout the paper, \( n \) denotes the actual number of processors, \( \Delta \) the maximum degree and \( D \) the diameter of the network.

**1.2. Main Results** Efficient protocols for \( k \)-point-to-point communication and broadcast are presented. Even though the protocols are probabilistic, it is shown how to acknowledge messages deterministically. The protocol for point-to-point communication requires a setup phase in which a BFS tree is constructed. The setup phase takes an average of \( O((n + D \log n) \log \Delta) \) time, after which \( k \) point-to-point communications require an average of \( O((k + D) \log \Delta) \) time. Therefore, the network allows a new communication every \( O(\log \Delta) \) time slots.

After the setup phase \( k \) broadcasts require an average of \( O((k + D) \log \Delta \log n) \) time. Hence the average throughput of the network is a broadcast every \( O(\log \Delta \log n) \) time slots.

Both protocols pipeline the messages along the edges of a BFS tree. They are always successful on the graph spanned by this tree. Their probabilistic behavior refers only to the running time. The performance analysis of both protocols constitutes a new application of queueing theory.

The main contributions of the current work is in the use of the BFS tree to pipeline the messages in the network, the acknowledgement mechanism and the usage of Queueing Theory for the analysis of the collection protocol. We feel that the last two contributions can be used in other applications.

**1.3. Previous Work** Chlamtac and Kutten [CK85] showed that, given a network and a designated source, finding an optimal broadcast schedule (i.e., broadcasting schedule which uses the minimum number of time-slots) is NP-Hard. Chlamtac and Weinstein [CW87] presented a polynomial-time (centralized) algorithm for constructing a broadcast schedule which uses \( O(D \log^2 n) \) time-slots. This centralized algorithm can be implemented in a distributed system assuming the availability of special control channels, but the number of control messages sent may be quadratic in the number of nodes of the network [We87].

Bar-Yehuda et al. [BGI87] described a randomized single-source broadcast protocol. To ensure that with probability \( 1 - \epsilon \) all nodes receive the message the protocol requires an average of \( O((D + \log \frac{n}{\epsilon}) \log \Delta) \) time slots. For \( D = 2 \), they have also shown a \( \Omega(n) \) lower bound for deterministic protocols. Thus, for this problem there exist randomized protocols that are much more efficient than any deterministic one. For \( D = 2 \), Alon et al. [ABLP88] showed an \( \Omega(\log^2 n) \) lower bound, which is close to the upper bound of [BGI87].

In [BGI88] Bar-Yehuda et al. discuss several models of radio communication and show how to detect conflicts and simulate a single hop network. Thus they show how to use protocols designed for the ETHERNET in the current network [C79, Ga85].
1.4. Protocol Outline. Both protocols depend on the existence of a BFS tree of the graph which is constructed in a setup phase. During the setup phase, a leader is chosen. Once the leader is chosen, it initiates the construction of a BFS tree whose root is the leader. For this purpose the protocols of [BG187] and [BG188] are used. The setup phase is conducted only once, after which any series of point-to-point transmissions or broadcasts might be performed.

The broadcast process is reactive (continuous), it is invoked whenever a source originates a message to broadcast. It consists of two subprotocols: collection — sending the messages from the sources to the root of the BFS tree, and distribution — sending the messages from the root to all the processors of the network.

The point-to-point transmission is also reactive. It is invoked whenever a processor wishes to send a message. A message from node \( u \) to \( v \) travels first up the tree. Once the message reaches a common ancestor of \( u \) and \( v \) it continues downwards towards \( v \). The protocols for both directions are very similar to the collection protocol and are fully described in section 5.

Since both processes are reactive, it is not possible to wait until all the messages have finished to travel upwards, and only then start their journey downwards. Therefore, the protocols for both directions are conducted concurrently, either by using separate channels or by multiplexing: odd time slots are dedicated to the upward traffic (collection) and the even ones to the downwards traffic. We shall not elaborate further and assume separate channels.

All our protocols make use of a basic protocol, \( \text{Decay} \) [BG187] for passing messages from one layer to the next. In the sequel we use the term \textit{send} whenever \( \text{Decay} \) is used.

```
procedure \text{Decay}(m);
repeat at most \( 2\log \Delta \) times
transmit \( m \) to all neighbors;
\text{flip coin } \in \{0,1\}
until \text{coin} = 0.
```

\text{Decay} is a probabilistic protocol, with the following properties:

1. It lasts \( 2\log \Delta \) time slots.

2. If several neighbors of a node \( v \) use \text{Decay} to send messages then with probability greater than \( \frac{1}{2} \) the node \( v \) receives one of the messages.

Several of our protocols require that all successfully sent messages be acknowledged. We show that even if there is positive probability that a message is not received, every message that has actually been received is acknowledged with certainty. The overhead of the acknowledgement mechanism is minimal: it slows down the protocol by a factor of 2. As a result the point-to-point transmission is always successful on the graph spanned by the BFS
1.5. Organization Section 2 describes the setup phase. It relies on previous work we only show how to modify it for our needs. All the other results are entirely new. Section 3 describes the acknowledgement mechanism, Section 4 the collection protocol and its analysis, Section 5 the point-to-point transmission protocol and Section 6 the distribution protocol. An application, ranking, is described in Section 7. Concluding remarks appear in Section 8.

2. The Setup Phase

Our protocols require the existence of a basic communication subnetwork. This network consists of a leader which is a root of a BFS tree. Bar-Yehuda et al. [BGI88] described how to find a leader in \( O((\log \log n)(D + \log R))\log \Delta \) time.

In [BGI87] Bar-Yehuda et al. describe how to find a BFS tree. Their algorithm requires \( O(D \log \Delta \log \frac{R}{e}) \) time slots and succeeds with probability \( 1 - e \). Since we have assumed that all the IDs are distinct and \( n \) is known to all the nodes, the leader election and BFS can be modified so that they always succeed, only the running time is probabilistic:

First, choose \( e = 1/n \), thus with probability \( \geq 1 - n^{-1} \) the leader election and the BFS protocol succeed. To make sure that the protocol always succeeds, when joining the tree each node sends a message to the root using the collection protocol of Section 4 below. This protocol only uses already constructed edges of the BFS tree, always succeeds and requires an average of \( O(n \log \Delta) \) time slots to send all these messages. If the root does not receive all the messages by twice the expected time, it reinvokes the setup phase. Since the probability of reinvocation is less than \( 1/2 \), the entire modified setup protocol lasts \( O(D \log \Delta \log n) \) time slots on the average.

2.1. Preventing collisions from different levels. An advantage of the BFS tree is that a collision at a node \( v \) at level \( i \) can occur only by messages sent from levels \( i-1 \), \( i \) and \( i+1 \). Collisions of messages sent from different levels are prevented by using time multiplexing: We require that a node at level \( i \) transmits a message at time slot \( t \) only if \( t = i \mod 3 \). This increases the duration of our protocols by a factor of 3. Henceforth, we assume that this mechanism has been built into all our protocols.


The protocols of sections 4 and 5 use messages which are each destined to a single processor. These protocols require that every message be acknowledged. We now show how to conduct acknowledgements deterministically. The odd time slots are dedicated to the original protocol and the even ones to acknowledgements. Namely, every
node that receives a message sends an acknowledgement on the next time slot.

The next theorem shows the correctness of this protocol. The theorem depends on the fact that each message has a unique destination and that the destinations of different messages successfully received at the same time slot are distinct.

**Theorem 3.1:** Let \( v \) be a node that received a message from node \( u \) using the above protocol, then \( u \) receives an acknowledgement.

**Proof:** Suppose that \( v \) received the message from \( u \) at time slot \( t \) and that \( u \) did not receive the acknowledgement. According to the protocol, \( v \) sent an acknowledgement at time slot \( t+1 \). Since \( u \) did not receive the acknowledgement there must have been a conflict at \( u \), i.e., at time slot \( t \) another node \( v' \) connected to \( u \) also sent an acknowledgement (see Figure 1).

![Figure 1](image)

According to the protocol, \( v' \) would not send an acknowledgement unless it received a message destined to it at time slot \( t \).

However, since the message sent by \( u \) was destined to \( v \neq v' \) and \( v' \) acknowledges only messages destined to it, \( v' \) received its message from a node \( u' \neq u \). Therefore, at time slot \( t \) both \( u \) and \( u' \) sent messages, and since \( v' \) is connected to both of them, a conflict occurred at \( v' \) and \( v' \) did not successfully receive any message. This contradicts the assumption that \( v' \) successfully received a message at time slot \( t \).

4. Collection

The purpose of the collection protocol is to send messages from the sources to the root of the BFS tree. Since no source knows the number and IDs of the other sources this is done concurrently and independently by all of them.

Messages are sent, using *Decay*, via the BFS tree from BFS-children to their parents. To each message we append the ID of the node \( v \) which sent the message and the ID of \( v \)'s BFS-parent. This information enables a node to figure out whether the message was sent by its BFS-child, by its BFS-parent or by another node. The nodes will make use of this information and we shall omit the details of this.
4.1. The collection protocol  Every node has a buffer of unacknowledged messages. Initially, all buffers except those of the sources are empty. The protocol proceeds in phases. In the odd time slots of each phase all nodes whose buffer is not empty execute $Decay$ to send a message from its buffer to its BFS-parent. The even time slots are dedicated to acknowledgements as explained in Section 3. Every such message is resent until an acknowledgement is received. Whereupon, it is removed from the sender’s buffer. When a message is received it is put on its receiver’s buffer. Since the acknowledgement occurs immediately after sending, messages exists on exactly one buffer and proceed from child to parent.

4.2. Analysis of the collection protocol  We first estimate how fast messages move from level to level.

Theorem 4.1  Let $i \geq 1$ be a level containing messages at the beginning of a phase. There is probability $\gamma = e^{-1}(1-e^{-1}) = 0.2325$ that during the phase a message from level $i$ is successfully received by its BFS parent.

Proof sketch: A phase consists of a single invocation of $Decay$. At any given time, the nodes which still want to transmit are called live. At each time slot each live processor first transmits and then with probability $l$ dies. Therefore, on the average, half of the live nodes die at each time slot.

Let $TRY_i$ be the set of nodes of level $i$ who are live at time $t=0$. Consider two cases:

Case 1, $|TRY_i| \leq \Delta$: The analysis of $Decay$ [BG187] implies that with probability $\geq \frac{1}{2}$ there exists a time slot with exactly one live processor $u$ of $TRY_i$. Hence, the BFS parent of $u$ receives $u$’s message.

Case 2, $|TRY_i| > \Delta$: The probability that a node $v \in TRY_i$ is live at time $t_0 = \log \Delta$ is $1/\Delta$. The probability that $TRY_i$ contains a live node at time $t_0$ is $1 - (1 - \frac{1}{\Delta})^{\Delta} \geq 1 - (1 - \frac{1}{\Delta})^\Delta \geq 1 - e^{-1}$.

Let $u$ be such a node, $w$ be its BFS-parent and $S = \{v \in TRY_i \mid (v,w) \in E \setminus \{u\}\}$ (the transmitting neighbors of $w$ not including $u$). If at time $t_0$, $S$ contains no live nodes, then $w$ receives $u$’s message. The existence of a live node in $S$ is independent of the behavior of $u$. The probability that at time $t_0$ $S$ contains no live node is $(1 - \frac{1}{\Delta})^{\Delta} \geq \left(1 - \frac{1}{\Delta}\right)^{\Delta-1}$. Since $\Delta \leq \Delta$ the above probability is $\geq e^{-1}$. Therefore, the probability of a successful transmission is $\geq (1 - e^{-1}) \times e^{-1}$.

We now use Queueing Theory to analyze the performance of the collection protocol. To that end, we model each level of the graph (except the root) as a server (see Figure 2). Thus we have $D$ servers connected in series,
with the output of the \( i \)th server being the input the \((i-1)\)st. Theorem 4.1 shows that if there are customers (messages) at server (level) \( i \), at some phase then with probability \( \geq \gamma \) a customer is served and moved to server \((i-1)\) during that phase.

![Figure 2 - The Queueing Model](image)

In our application at time \( t=0 \) the messages are distributed over all levels. The processing time of our system is less than that of a system in which all messages originate at level \( D \) and instead of being already in the system at time \( t=0 \), their arrival process is Bernoulli with parameter \( \lambda \).

We first analyze server \( D \). It is a discrete \( M/M/1 \) queue \([K75]\) with parameters \( \lambda \) and \( \mu \): Its birth process (arrival of customers) is Bernoulli with probability \( \lambda \), while the death process (serving customers) is Bernoulli with probability \( \mu \). Following Burke \([B56]\), Hsu and Burke \([HB76]\) analyzed the problem when \( \lambda < \mu \).

**Theorem 4.2**: \([HB76]\) Consider the departure process of the above queue (with \( \lambda < \mu \)), i.e., let \( \delta(t) = 1 \) if at time \( t \) a customer was processed, and \( \delta(t) = 0 \) otherwise. Then \( \delta \) converges to a Bernoulli process with parameter \( \lambda \).

They also showed that \( P_j(t) \) is the probability that at time \( t \) the length of the queue is \( j \) approaches a limit \( p_j \) and

\[
p_0 = 1 - \frac{\lambda}{\mu}, \quad p_1 = \frac{\lambda}{(1-\lambda)\mu} p_0, \quad p_j = \left[ \frac{\lambda(1-\lambda)}{\mu(1-\lambda)} \right]^{j-1} p_1.
\]

Thus the expected queue length is \( \bar{N} = \sum_{j=0}^{\infty} j p_j = \frac{\lambda(1-\lambda)}{\mu - \lambda} \).

By Little’s result \([K75\ p. 17]\), the average time in the queue is, \( \bar{t} = \frac{\bar{N}}{\lambda} = \frac{1 - \lambda}{\mu - \lambda} \).

The major observation is that since the output of the \( i \)th server is the input to the \((i-1)\)st, the input to all servers is Bernoulli with parameter \( \lambda \). Using this observation we get,
Theorem 4.3: The expected number of phases required for k messages to reach the root is at most $2(1 + \sqrt{1-\gamma})\gamma^{-1}(k + D)$.

**Proof:** Theorem 4.2 holds asymptotically. Here we have only a finite number of messages. However, the asymptotic results can be used (i.e., $t \to \infty$) since the queue sizes and the completion time are nondecreasing functions of $t$. Hence the limiting case is an upper bound.

The expected time for the k th message to arrive equals the expected time until it enters the queueing model $= k/\lambda$, plus the time to pass through the D servers $= \frac{1-\lambda}{\mu-\lambda} D$, i.e., a total of $\frac{k}{\lambda} + \frac{1-\lambda}{\mu-\lambda} D$.

Substituting $\lambda = 1 - \sqrt{1-\gamma}$ and $\mu = \gamma$ satisfies $\lambda < \mu$ and yields the desired result. $\Box$

Since each phase lasts twice the time of Decay and $\gamma = e^{-1}(1-e^{-1})$, we get

**Corollary:** The expected number of time slots for k messages to reach the root is bounded by $32.27(k+D)\log D$.

This constant can be improved using the techniques of [Ho87].

5. Point to Point Transmission

As mentioned before, this protocol consists of two subprotocols: the upward direction subprotocol from the initiator of the message $u$ to a common ancestor $w$ of $u$ and the destination $v$ and the downward direction subprotocol from $w$ to $v$.

5.1. The upward subprotocol This protocol is essentially identical to the collection protocol, except that the message does not go all the way to the root but only to the least common ancestor of the originator and the destination which are included in the message. When the message reaches a BFS tree ancestor of the destination, the downward protocol is invoked. To this end, each node should hold list of its BFS descendants.

The descendant information can be found once, right after the BFS tree is constructed: As soon as a node joins the BFS tree, it sends its ID to the root via all its ancestors. During the BFS protocol, each node can record its parent, thus to send a message to the root, it is sufficient to send it to that parent and ask it to send the message further. Whenever a node receives such a message from one of its children, it adds the ID of the originator of the message to the list of IDs of descendants. Conveying all the descendant information requires the collection of $n-1$ messages, i.e., $O((n-1+D)\log D) = O(n \log n)$ time.
5.2. The downward subprotocol  This protocol is also very similar to the collection protocol. The messages are prepended with the ID of their final destination. Every node sends messages destined to its BFS children and keeps a buffer of unacknowledged downgoing messages. Here we also use Decay. On each phase a message is sent, according to that protocol. Messages are resent until an acknowledgement is received. A node $w$ receiving a message destined to $u$ processes it only if $u$ is a BFS-tree descendant of $w$. To process a message, $w$ acknowledges it and if $w \neq u$ it is put on $w$'s downgoing buffer.

5.3. Performance analysis  The setup phase requires $O(n \log \Delta)$ time after which passing $k$ messages requires an average of $O((k+D) \log \Delta)$ time. When $k \rightarrow \infty$ the average time per message is $O(\log \Delta)$.

6. Broadcast

To broadcast a message a node first sends the message to the root using the collection subprotocol of Section 4. Then the message is sent to all the nodes of the network using the distribution subprotocol to be described below.

In the distribution protocol every message has several destinations, therefore, the acknowledgement mechanism of Section 3 can no longer be used. In principle the message can be sent using the BFS protocol. However, each message would require $2D \log \Delta \log n$ time to reach all the nodes with probability $1-\epsilon$. A better idea is to use pipelining: Send the $i+1$-st message before the $i$-th one reaches its destination:

The protocol consists of superphases each consisting of $4 \log \Delta \log n$ time slots. At superphase $t$ the root sends the $t$-th message and all the nodes of level $i$ repeatedly send the $t-i$-th message (using the Decay protocol $2 \log n$ times).

Let $v$ be a node of level $i$ and $t$ a superphase in which the nodes of level $i-1$ send the message $m$. By property (2) of Decay, in each invocation of Decay, there is probability $\geq \frac{1}{2}$ that $v$ receive a message. Since there can be no interference by messages sent from different levels (Section 2.2), if $v$ receives any message it must be from level $i-1$ and since all the nodes of that level send the same message, with probability $\geq \frac{1}{2}$ $v$ receives the message $m$. Since a superphase consists of $2 \log n$ invocations of Decay there is probability $\geq 1 - 1/n^2$ that $v$ receives $m$ during the superphase. The probability, that $m$ is passed successfully to all the nodes of the network is $\geq 1 - 1/n$.

For each message the above protocol may fail with finite probability. If the number of messages is unbounded, then eventually the protocol will fail. This failure can be prevented by changing the protocol as follows:

The root appends consecutive numbers to the messages. Every node $v$ examines these numbers and when $v$ encounters a gap it realizes that it did not receive a message. Thereupon, $v$ sends a message to the root requesting it to resend the missing message.
Since on the average no more than $1/n$ of the messages will be resent, the extra load on the network is a factor of
\[
\sum_{i=0}^{n} \frac{1}{n^i} = \frac{n}{n-1}.
\]
Moreover, the time spent in each layer is $O(\log \Delta \log n)$, thus the effective rate in which message leave the root is $O\left(\frac{n}{n-1} \log \Delta \log n\right) = O(\log \Delta \log n)$. Also, each message requires an average of $O(D \log \Delta \log n)$ time slots to reach all the nodes.

The previous change causes another problem, the message numbers are unbounded. An additional change can correct this problem. The messages are numbered mod $n^2$. After message number $n^2$ is received each node sends an acknowledgement to the root. The expected time that all these messages reach the root is $O((n+D) \log \Delta)$. Thus the expected time that the all the acknowledgements reach the root is $O((D \log \Delta + n) \log \Delta)$. Let $c$ be the implied constant in the above expression. If the root does not receive acknowledgements from all the nodes by $2c(D \log \Delta + n) \log \Delta$ time slots after it sent the $n^2$th message it resends the missing messages. It can be shown that the probability that the $n^2$th message has to be resent is less than $\frac{1}{2}$, thus this last correction increase the load of the system by at most a factor of 2.

7. Ranking

Our protocols can be used for additional problems, such as ranking in expected time $O(n \log n \log \Delta)$:

The problem

Given $n$ processors with distinct IDs $id_1, \ldots, id_n$, renumber the processors, $id'_1, \ldots, id'_n$ such that $1 \leq id'_1 \leq n$ and $id'_i < id'_j$ if and only if $id_i < id_j$.

The protocol

Use point-to-point communication to send all the IDs to the root. It calculates the destination of each of the new IDs and sends them to the nodes. Thus there is a total of $2n-2$ messages, which require $O(n \log \Delta)$ time (not including the setup costs of Section 2).

8. Remarks and Open Problems

(1) If $n$ is not known but only an upper bound $N$, we can still find a BFS tree with probability $1-\varepsilon$ in expected time $O(D \log \frac{N}{\varepsilon} \log \Delta)$. This setup time is sufficient for $k$-broadcast. However, point-to-point transmissions still requires $O(n + D \log \frac{N}{\varepsilon} \log \Delta)$ time to acquire the descendant information.

(2) If there are no IDs then the processors can randomly choose sufficiently long IDs such that with probability $1-\varepsilon$ all the IDs are distinct.
(3) Suppose that we change the model such that in case of a conflict the receiver may get one of the messages. In this model our deterministic acknowledgement mechanism is no longer valid. A more complicated, less reliable and slower protocol exists also for this case.

(4) In some "real life" situations processors can detect that a conflict occurred. We have not postulate this ability since we do not know how to use it.

(5) Our protocols route messages through a spanning tree causing congestion at the root. Are there efficient communication protocols that avoid this problem?

ACKNOWLEDGEMENTS

It is a pleasure to thank Shay Kutten for helpful discussions.