Multiple Communication in Multi-Hop Radio Networks †

Reuven Bar-Yehuda[1]
Department of Computer Science

Amos Israeli[2]
Department of Electrical Engineering

Alon Itai[3]
Department of Computer Science

Technion - Israel Institute of Technology,
Haifa, 32000, ISRAEL.

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 Δ 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 the setup, k point-to-point transmissions require $O((k+D)\log\Delta)$ time on the average. Therefore the network allows a new transmission every $O(\log\Delta)$ time slots. Also, 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.

Keywords: Radio networks, broadcast, point-to-point routing, distributed algorithms, average case analysis, Queueing Theory.

---

† A preliminary version of this paper was presented in the Symposium on Principles of Distributed Computing, Edmonton, Canada, 1989.

[1] Partially supported by Technion V.P.R. Fund - Albert Einstein Research Fund.
[2] Partially supported by the Technion V.P.R. Fund, New York Metropolitan Fund and the Japanese TS Research Fund.
[3] Partially supported by Technion V.P.R. Fund.

1. Introduction

A *radio network* is a network of processors which communicate using radio. An important feature of radio communication is that if a receiver is in the range of two or more transmitting stations then due to interferences some messages might not be received. 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 through intermediate stations.

Most real life radio networks for data communication are quite limited. In fact, most such networks are single-hop and most 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 [3], 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 [3], 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 station 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 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 and within line of sight of each other). The processors 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 Δ 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 [4]).

Throughout the paper, n denotes the actual number of processors, Δ the maximum degree and D the diameter of the network.

1.2 Main Results Efficient protocols for k-point-to-point communication and k-broadcast are presented. Even though the protocols are probabilistic, it is shown how to acknowledge messages deterministically. 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 the setup, k point-to-point transmissions require $O((k+D)\log\Delta)$ time on the average. Therefore the network allows a new transmission in every sequence of $O(\log\Delta)$ time slots. Also, k broadcasts require an average of $O((k+D)\log\Delta\log n)$ time. Hence the average throughput of the network is a broadcast in every sequence of $O(\log\Delta\log n)$ 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.

1.3 Previous Work Chlamtac and Kutten [7] 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. They also routed messages through a (not necessarily BFS) tree, and discussed "implicit acknowledgements". Their acknowledgments are conducted in the absence of conflicts, which is achieved at the cost of increasing the time of a single point-to-point communication to $O(D\Delta)$.

Chlamtac and Weinstein [8] 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 [16].

In a different context, Y. Birk [2] independently discovered an acknowledgement mechanism similar to ours.

Bar-Yehuda et al. [3] described a randomized single-source broadcast protocol. To ensure that with probability $1-\varepsilon$ all nodes receive the message the protocol requires $O((D+\log\frac{n}{\varepsilon})\log\Delta)$ time slots. For $D=2$, they have also shown an $\Omega(n)$ 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. [1] showed an $\Omega(\log^2 n)$ lower bound, which matches the upper bound of [8] and [3].

In [4] 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 a multi-hop network [6, 1].

1.4 Protocol Outline Both protocols depend on the existence of a BFS tree of the graph which is constructed in a setup phase. In the beginning of the setup part, 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 [3] and [4] 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 protocols 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, in both protocols the collection and distribution subprotocols are conducted concurrently, either by using separate channels or by multiplexing: the 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, *Decay* [3] for passing messages from one layer to the next. In the sequel we use the term *send* whenever *Decay* is used.

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

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 *Decay* to send messages then with probability greater than ½ the node v receives one of the messages.

Several of our protocols require that all successfully sent messages be acknowledged. We show that though 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 tree.

1.5 Organization Section 2 describes the setup phase. Since 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. [4] described how to find a leader in $O((\log\log n)\cdot(D + \log\frac{n}{\varepsilon})\log\Delta)$ time.

In [3] Bar-Yehuda et al. describe how to find a BFS tree. Their algorithm assumes that all nodes wake up at time 0. It requires $O(D \log\Delta \log \frac{n}{\varepsilon})$ time slots and succeeds with probability $1-\varepsilon$. 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 random:

First, choose $\varepsilon = 1/n$, thus with probability $\geq 1-n^{-1}$ the leader election and the BFS protocol succeed. To ensure 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, the algorithm is aborted and the entire setup phase is reinvoked. Note that since all nodes know when the invocation should terminate, different invocations by the same processor cannot exist concurrently.

Since the probability of reinvocation is less than ½, the entire modified setup protocol lasts $O((n+D\log n)\log\Delta)$ 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 $t \equiv i \bmod 3$. 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.

3. The Acknowledgement Protocol.

The protocols of sections 4 and 5 use messages which are each designated 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 $t+1$ another node, v', connected to u also sent an acknowledgement (see Figure 1). According to the protocol, v' would not send an acknowledgement unless it received a message designated to it at time slot t.

However, since the message sent by u was designated to $v \neq v'$ and v' acknowledges only messages designated 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' (at time slot t) 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

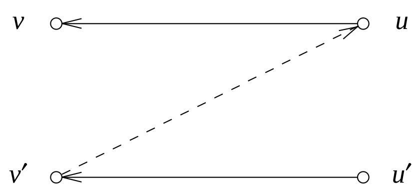

Figure 1

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 every node whose buffer is not empty executes *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 exist on exactly one buffer and proceed from child to parent.

4.2. Analysis of the collection protocol

4.2.1 Transmission between adjacent levels

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 $\geq \mu \stackrel{\text{def}}{=} e^{-1}(1-e^{-1}) \approx 0.2325$ that during the phase a message from level i is successfully received by its BFS parent.*

Before sketching the proof, note the difference between this theorem and property (2) of *Decay*. Suppose node u is sending a message to its neighbor v and node u' to its neighbor v'. If u' is connected also to v and u' to v, then property (2) is satisfied even when v gets the message of u' or v' gets that of u. In contrast, here we insist that each message arrives at its correct destination.

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 ½ *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* [3] implies that with probability ≥ ½ 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 $t_0 \overset{\text{def}}{=} \log\Delta$ is $1/\Delta$. The probability that TRY_i contains a live node at time t_0 is $1 - (1 - \frac{1}{\Delta})^{|TRY_i|} \geq 1 - (1 - \frac{1}{\Delta})^{\Delta} \geq 1 - e^{-1}$.

Let u be the first such node in some fixed arbitrary order, w 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 at least $(1 - \frac{1}{\Delta})^{|S|} \geq \left((1 - \frac{1}{\Delta})^{\Delta-1} \right)^{|S|/(\Delta-1)}$.

Since $|S| \leq \Delta - 1$ the above probability is $\geq e^{-1}$. Therefore, the probability of a successful transmission is $\geq$

$Prob(\ TRY_i$ contains a live node at time $t_0) \times Prob($ at time t_0 all nodes in S dead$) \geq (1 - e^{-1}) \times e^{-1}$. □

4.2.2 Outline of the analysis. In the remainder of the section we shall show an upper bound on expected completion time of the collection protocol. We shall go through a series of models and show that when moving from one model to the next the expected completion time can only increase. In this subsection we describe the models, a formal proof follows in subsequent subsections.

The first model is the previously discussed radio network which contains a BFS-tree of depth D and k messages arbitrarily placed on the nodes of the tree. In Theorem 4.1 we have shown that there is probability $\geq \mu = e^{-1}(1 - e^{-1})$ that among all the messages placed in the nodes of level i at least one moves to level $i-1$.

The second model consists of a path of $D+1$ nodes. All the messages in the ith level of the previous model reside in node i of the path. The root of the tree is now node 0. We also stipulate that in a single step at most one message can move from node i to node $i-1$ and the probability for that is exactly μ.

In the third model the messages are not already present in the system at time = 0, but their arrival is a Bernoulli event with parameter $\lambda < \mu$, i.e., for every time t there is probability λ that a new message appears at node D.

The last model we introduce (model 4) is identical to the third but we assume that it is already in steady state in the sense of Queueing Theory (see [14]) and we define the expected completion time to be the expected time for *k additional* messages to arrive at node *D* and then proceed to the root (node 0).

4.2.3 A tandem queue of Bernoulli servers.

We now use Queueing Theory to analyze the performance of model 4: The model consists of *D* servers connected in series, with the output of the *i*th server being the input to the $i-1$st. We first analyze the behavior of a single level.

A *Bernoulli server with parameter* μ is a discrete server (i.e., it operates in discrete time steps) such that if at any time step the queue of incoming customers is nonempty then with probability μ during that time step exactly one customer is served (removed from the incoming queue and placed on the outgoing queue). The *arrival rate* λ is the probability that a new customer (i.e., message) appears in the incoming queue during a phase. The *departure process* is the process by which customers are served by the server. Following Burke [*B* 56], Hsu and Burke [*HB* 76] analyzed the behavior of the departure process when $\lambda < \mu$.

Theorem 4.2: [*HB* 76] *Consider the departure process of the above server (with* $\lambda < \mu$*), i.e., let* $\delta(t) = 1$ *if at time t a customer was processed, and* $\delta(t) = 0$ *otherwise. Then* δ *converges to a Bernoulli process with parameter* λ.

They also showed that the probability that at time *t* the length of the queue is *j* approaches a limit p_j and

$$p_0 = 1 - \frac{\lambda}{\mu}, \qquad p_1 = \frac{\lambda}{(1-\lambda)\mu} p_0 \,, \qquad p_j = \left(\frac{\lambda(1-\mu)}{\mu(1-\lambda)}\right)^{j-1} p_1 . \tag{*}$$

Thus the expected queue length is $\overline{N} = \sum_{j \geq 0} j p_j = \frac{\lambda(1-\lambda)}{\mu - \lambda}$. By Little’s result [*K* 75 p. 17], in steady state the average time in the queue is, $E(T) = \frac{\overline{N}}{\lambda} = \frac{1-\lambda}{\mu-\lambda}$.

We now return to model 4, a steady state network of *D* Bernoulli processors connected in series each with parameter μ and the arrival rate to the *D*th server is $\lambda<\mu$. 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 λ. Using this observation we get the following theorem

Theorem 4.3 *The expected completion time of model 4 is* $\frac{k}{\lambda} + \frac{1-\lambda}{\mu-\lambda}D$.

Proof: Suppose that at time t_0 the queueing system is in steady state and a message m_0 arrives. Let T denote the time the message spends in the queues. Consider the k messages $m_1, \ldots, m_k$ preceding message m_0. Let X_i denote the interarrival time between message m_i and the next message, m_{i-1}. Define

$$Q_k = T + X_1 + X_2 + \cdots + X_k .$$

Q_k is the time for k messages to pass through the queueing system of model 4. The expected time is

$$E(Q_k) = E(T) + E(X_1) + E(X_2) + \cdots + E(X_k).$$

The theorem follows from the fact that $E(X_i) = \frac{1}{\lambda}$ and from the previous discussion which showed that $E(T) = \frac{1-\lambda}{\mu-\lambda}D$. □

In Theorem 4.15 we shall show that the expected completion time of model 1 is less than or equal to that of model 4. Thus the performance of model 4 constitutes an upper bound for the radio network. Substituting $\lambda = 1 - \sqrt{1-\mu}$ satisfies $\lambda<\mu$ and yields that the expected number of phases required for k messages to reach the root is at most $2(1+\sqrt{1-\mu})\mu^{-1}(k+D)$. Since each phase lasts twice the time of *Decay* and $\mu = e^{-1}(1-e^{-1})$, we get the following theorem

Theorem 4.4 *The expected number of time slots for k messages to reach the root is bounded by* $32.27(k+D)\log\Delta$.

This constant can be improved using the techniques of [11].

4.2.4 The expected completion time of model 3 is not greater than model 4.

The difference between model 3 and 4 is that model 3 stipulates that initially all queues are empty, whereas in model 4 the queue in each node has reached steady state, in particular there is nonnegative probability that the queues are not empty. In this subsection we prove the intuitively clear point that adding messages to the queues can only increase the expected completion time.

Consider partitions of messages between the levels, i.e., $(D+1)$-vectors $\mathbf{a} = (a_1, \ldots, a_{D+1})$ such that $a_i \geq 0$. A *move vector* is a $(D+1)$-vector of nonnegative integers, $\mathbf{m} = (m_1, \ldots, m_{D+1})$. Partition

$\mathbf{a}' = Move(\mathbf{a},\mathbf{m})$ is obtained by moving m_i messages from level i to level $i-1$, (if $m_i > a_i$ then only a_i messages are moved). Formally, the number of messages moved from level i is $\delta_i = \min(a_i, m_i)$, $i=1,\ldots,D$, $\delta_{D+1} = m_{D+1}$. Therefore, $a_i' = a_i - \delta_i + \delta_{i+1}$.

A *move sequence* is an infinite series $M = (\mathbf{m}^1, \mathbf{m}^2, \cdots)$ of move vectors. $Move^*(\mathbf{a}, M, t)$ is the result of making t moves according to M, i.e.,

$$Move^*(\mathbf{a}, M, 0) = \mathbf{a}$$
$$Move^*(\mathbf{a}, M, t+1) = Move(Move^*(\mathbf{a}, M, t), \mathbf{m}^{t+1}).$$

Define a partial order $\leq$ between partitions, such that $\mathbf{a} \leq \mathbf{b}$ if and only if there exists a move sequence M and an integer t such that $\mathbf{a} = Move^*(\mathbf{b}, M, t)$. (Also, $\mathbf{a} < \mathbf{b}$ if $\mathbf{a} \leq \mathbf{b}$ and $\mathbf{a} \neq \mathbf{b}$.)

A move vector $\mathbf{m}$ is a *singleton* if exactly one of its components is 1 and all the other components are zero; the singleton whose i-th component is 1 is denoted $\mathbf{e}_i$. The following lemma shows that we can simulate any move vector by a move sequence of lexicographically nonincreasing singletons.

Lemma 4.5: *For every move vector* $\mathbf{m}$ *there is a singleton move sequence* $E_{\mathbf{m}}$ *such that for every* $\mathbf{a}$, $Move(\mathbf{a},\mathbf{m}) = Move^*(\mathbf{a}, E_{\mathbf{m}}, \sum_{i=1}^{D} m_i)$.

Proof: Let $E_{\mathbf{m}} = (\mathbf{e}^1_{\mathbf{m}}, \mathbf{e}^2_{\mathbf{m}}, \cdots)$ such that $\mathbf{e}^t_{\mathbf{m}} = \mathbf{e}_j$, where j is the first nonzero component of $\mathbf{m} - \sum_{i=1}^{t-1} \mathbf{e}^{t-1}_{\mathbf{m}}$. □

Corollary 4.6: $\mathbf{a} \leq \mathbf{b}$ *if and only if there exist an integer t and a move sequence* E *consisting only of singletons such that* $\mathbf{a} = Move^*(\mathbf{b}, E, t)$.

Lemma 4.7: *If* $\mathbf{a} \leq \mathbf{b}$ *then for any move vector* $\mathbf{m}$, $Move(\mathbf{a},\mathbf{m}) \leq Move(\mathbf{b},\mathbf{m})$.

Proof: Lemma 4.5 and Corollary 4.6 imply that it suffices to prove the lemma for $\mathbf{m} = \mathbf{e}_j$ and $\mathbf{a} = Move(\mathbf{b}, \mathbf{e}_i)$. If $i \neq j+1$ then $Move(\mathbf{a}, \mathbf{e}_j) = Move(Move(\mathbf{b}, \mathbf{e}_i), \mathbf{e}_j) = Move((\mathbf{b}, \mathbf{e}_j), \mathbf{e}_i)$, implying $Move(\mathbf{a}, \mathbf{e}_j) \leq Move(\mathbf{b}, \mathbf{e}_j)$.

Also if $b_i = 0$ then $\mathbf{a} = \mathbf{b}$. Thus we assume that $i = j+1$ and $b_i > 0$. If $b_j = 0$ then $\mathbf{b} = Move(\mathbf{b}, \mathbf{e}_j)$ and $Move(\mathbf{a}, \mathbf{e}_j) = Move(Move(\mathbf{b}, \mathbf{e}_{j+1}), \mathbf{e}_j)$. From the definition of $\leq$, $Move(Move(\mathbf{b}, \mathbf{e}_{j+1}), \mathbf{e}_j) \leq b = Move(\mathbf{b}, \mathbf{e}_j)$. Otherwise ($b_j \neq 0$), $Move(\mathbf{a}, \mathbf{e}_j) = Move(Move(\mathbf{b}, \mathbf{e}_{j+1}), \mathbf{e}_j) = Move(Move(\mathbf{b}, \mathbf{e}_j), \mathbf{e}_{j+1}) \leq Move(\mathbf{b}, \mathbf{e}_j)$. □

The *completion time* of a partition **a** w.r.t. a move sequence M is $T(\mathbf{a},\mathrm{M}) = \min\{t : Move(\mathbf{a},\mathrm{M},t) = (0,0,\ldots,0)\}$. (For some M's the completion time may be infinite.)

Lemma 4.8: *If* $\mathbf{a} \le \mathbf{b}$ *then for all* M, $T(\mathbf{a},\mathrm{M}) \le T(\mathbf{b},\mathrm{M})$.

Proof: Let **b** be the least partition for which there exists a move sequence M and a partition **a** such that $\mathbf{a}<\mathbf{b}$ while $T(\mathbf{a},\mathrm{M}) > T(\mathbf{b},\mathrm{M})$. Let M′ satisfy,

$$T(\mathbf{b},\mathrm{M}') = \mathrm{Min}\{T(\mathbf{b},\mathrm{M}) : T(\mathbf{b},\mathrm{M}) < T(\mathbf{b},\mathrm{M})\}.$$

Let $\mathrm{M}' = (\mathbf{m}^1,\mathbf{m}^2,\cdots)$. The minimalilty of M′ implies that $Move(\mathbf{b},\mathbf{m}^1) < \mathbf{b}$. Define $\mathrm{M}''=(\mathbf{m}^1,\mathbf{m}^2,\cdots)$. By Lemma 4.7 $Move(\mathbf{a},\mathbf{m}^1) \le Move(\mathbf{b},\mathbf{m}^1)$, thus by the minimality of **b**, $T(Move(\mathbf{a},\mathbf{m}^1),\mathrm{M}'') \le T(Move(\mathbf{b},\mathbf{m}^1),\mathrm{M}'')$. Hence,

$$T(\mathbf{a},\mathrm{M}') \le 1 + T(Move(\mathbf{a},\mathbf{m}^1),\mathrm{M}'') \le 1 + T(Move(\mathbf{b},\mathbf{m}^1),\mathrm{M}'') = T(\mathbf{b},\mathrm{M}'). \qquad \square$$

Consider an arbitrary probability distribution on the move sequences. In a tandem queue of Bernoulli servers with parameter μ and arrival rate λ, $P(m_i^t=1) = \mu$, $\quad i=1,\ldots,D$ and $P(m_{D+1}^t=1) = \lambda$.

Let $p_t(\mathbf{a})$ be the probability that $T(\mathbf{a},\mathrm{M}) = t$. The *average completion time* is $E(T(\mathbf{a})) = \sum_{t=1}^{\infty} t p_t(\mathbf{a}) = \sum_{t=1}^{\infty}\sum_{j=t}^{\infty} p_j(\mathbf{a})$.

Lemma 4.9 $\mathbf{a} \le \mathbf{b}$ *implies that* $E(T(\mathbf{a})) \le E(T(\mathbf{b}))$.

Proof: Let $\mathbf{a} \le \mathbf{b}$. Thus by Lemma 4.8,

$$\{\mathrm{M} : T(\mathbf{a},\mathrm{M}) \le t\} \supseteq \{\mathrm{M} : T(\mathbf{b},\mathrm{M}) \le t\}.$$

Taking probabilities

$$P(\{\mathrm{M} : T(\mathbf{a},\mathrm{M}) \le t\}) \ge P(\{\mathrm{M} : T(\mathbf{b},\mathrm{M}) \le t\}).$$

In other words,

$$\sum_{j=0}^{t} p_j(\mathbf{a}) \ge \sum_{j=0}^{t} p_j(\mathbf{b}).$$

The last condition defines that $T(\mathbf{a})$ is *stochastically greater than* $T(\mathbf{b})$ [15]. In this case,

$$E(T(\mathbf{a})) = \sum_{t=1}^{\infty}\sum_{j=t}^{\infty} p_j(\mathbf{a}) \le \sum_{t=1}^{\infty}\sum_{j=t}^{\infty} p_j(\mathbf{b}) = E(T(\mathbf{b})). \qquad \square$$

Lemma 4.10 *The expected completion time of model 3 is less than or equal to that of model 4.*

Proof: In model 4 the expected completion time depends both on the distribution of the initial partition and on the move sequences.

Let $\mathbf{a}=(a_1, \ldots, a_D, a_{D+1})$ be an initial partition. $a_1, \ldots, a_D$ are the lengths of the queues at the nodes, while $a_{D+1} = k$ since the completion time of model 4 is the time k additional messages appear at node D and reach the root.

In model 3 the initial partition is $\mathbf{k} = (0,0, \ldots, 0,k)$. $\mathbf{k} \le \mathbf{a}$ since for $M = ((a_1, \ldots, a_D, 0), (a_2, \ldots, a_D, 0, 0), \ldots, (a_D, 0, \ldots, 0), (0, \ldots, 0), \ldots)$, $\mathbf{k} = Move^*(\mathbf{a}, M, D)$. By Lemma 4.9, $E(T(\mathbf{k})) \le E(T(\mathbf{a}))$. Our result follows since this holds for every initial partition of model 4. $\square$

4.2.5 The expected completion time of model 2 is not greater than model 3.

Lemma 4.11 *The expected completion time of model 2 is less than or equal to that of model 3.*

Proof: In model 3 the initial partition is $\mathbf{k} = (0,0, \ldots, 0,k)$, while in model 2 it is $\mathbf{b} = (b_1, \ldots, b_D, 0)$, such that $k = \sum b_i$. Obviously, $\mathbf{b} \le \mathbf{k}$, so the result follows, as before from Lemma 4.9. $\square$

4.2.6 The expected completion time of model 2 is not smaller than model 1.

The difference between model 1 and 2 is in the move vectors. In model 2, $m_i^t \in \{0,1\}$ and $P(m_i^t = 0) = 1 - \mu$, while in model 1, m_i^t can assume any nonnegative integer value and $P(m_i^t = 0) = 1 - \mu$. The actual movements of the messages depend on the topology. To prove that the expected completion time of model 2 is not greater than that of model 1 we need to consider the effect of changing the move vectors.

A move vector $\mathbf{m} = (m_1, m_2, \ldots, m_{D+1})$ *dominates* the move vector $\tilde{\mathbf{m}} = (\tilde{m}_0, \tilde{m}_2, \ldots, \tilde{m}_{D+1})$ if for all i, $m_i \ge \tilde{m}_i$.

Lemma 4.12: *If* $\mathbf{m}$ *dominates* $\tilde{\mathbf{m}}$ *and* $\mathbf{a} \le \mathbf{b}$ *then* $Move(\mathbf{a},\mathbf{m}) \le Move(\mathbf{b},\tilde{\mathbf{m}})$.

Proof: Let $E_{\mathbf{m}}$ and $E_{\tilde{\mathbf{m}}}$ be the singleton move sequences corresponding to $\mathbf{m}$ and $\tilde{\mathbf{m}}$ (Lemma 4.5). Let t and $\tilde{t}$ be the respective lengths . Since $E_{\tilde{\mathbf{m}}}$ is a subsequence of $E_{\mathbf{m}}$, by repeated use of Lemma 4.7 we can show that $Move^*(\mathbf{a},E_{\mathbf{m}},t) \le Move^*(\mathbf{a},E_{\tilde{\mathbf{m}}},\tilde{t})$. Thus, by Lemma 4.5,

$$Move(\mathbf{a},\mathbf{m}) = Move^*(\mathbf{a},E_{\mathbf{m}},t) \le Move^*(\mathbf{a},E_{\tilde{\mathbf{m}}},\tilde{t})$$

By Lemma 4.7,

$$\le Move^*(\mathbf{b},E_{\tilde{\mathbf{m}}},\tilde{t}) = Move(\mathbf{b},\tilde{\mathbf{m}}). \qquad \square$$

A move sequence $M(\mathbf{m}^1,\mathbf{m}^2,\cdots)$ *dominates* $\tilde{M}(\tilde{\mathbf{m}}^1,\tilde{\mathbf{m}}^2,\cdots)$ if for all j, $\mathbf{m}^j$ dominates $\tilde{\mathbf{m}}^j$.

Lemma 4.13: *If* M *dominates* $\tilde{M}$ *and* $\mathbf{a} \le \mathbf{b}$ *then* $T(\mathbf{a},M) \le T(\mathbf{b},\tilde{M})$.

Proof: By induction on $T(\mathbf{a},M)$ and using lemma 4.12. $\square$

Lemma 4.14: *The expected completion time of model 1 is less than or equal to that of model 2.*

Proof: Consider an instance $\mathbf{a}^1,\mathbf{a}^2,\ldots,\mathbf{a}^T$ of model 1, i.e., the completion time is T and a_i^t is the number of messages at level i at time $t \le T$. Define the move sequence $M = (\mathbf{m}^1,\mathbf{m}^2,\cdots)$ as follows: m_i^t is the number of messages that moved from level i to level $i-1$ at time t. When $a_i^t > 0$ then by Theorem 4.1, $P(m_i^t \ge 1) \ge \mu$. However, when $a_i^t = 0$ then $m_i^t = 0$, so obviously, $P(m_i^t \ge 1) = 0$, violating the probabilistic assumptions of model 2. We therefore define the move sequence $\overline{M} = (\overline{\mathbf{m}}^1,\overline{\mathbf{m}}^2,\cdots)$: if $a_i^t > 0$ then $\overline{m}_i^t = m_i^t$, otherwise ($a_i^t = 0$), $\overline{m}_i^t = 1$. By Theorem 4.1 and the construction $P(\overline{m}_i^t \ge 1) \ge \mu$. Since M and $\overline{M}$ differ only when $a_i^t = 0$, for every $t \le T$, $Move^*(\mathbf{a},M,t) = Move^*(\mathbf{a},\overline{M},t)$. Thus the completion times $T(\mathbf{a},\overline{M}) = T(\mathbf{a},M) = T$.

Construct a move sequence $\tilde{M} = (\tilde{\mathbf{m}}^i,\ldots,\tilde{\mathbf{m}}^T,\cdots)$: ($\tilde{m}_i^t \in \{0,1\}$). If $\overline{m}_i^t \ge 1$ then $\tilde{m}_i^t = 1$ with probability $\dfrac{P(m_i^t \ge 1) - \mu}{P(m_i^t \ge 1)}$, otherwise $\tilde{m}_i^t = 0$. Since $\overline{M}$ dominates $\tilde{M}$, by Lemma 4.13, the completion time of $\overline{M}$ is less than or equal to that of $\tilde{M}$.

By the construction of $\tilde{M}$,

$$P(\tilde{m}_i^t = 0) = P(\overline{m}_i^t = 0) + P(\overline{m}_i^t > 0 \text{ and } \tilde{m}_i^t = 0)$$

$$= P(\overline{m}_i^t = 0) + P(\overline{m}_i^t > 0)\frac{P(\overline{m}_i^t > 0) - \mu}{P(\overline{m}_i^t > 0)} = (1 - P(\overline{m}_i^t > 0)) + (P(\overline{m}_i^t > 0) - \mu) = 1 - \mu.$$

Thus, the distribution of $\tilde{M}$ is identical to the distribution of the move sequences of model 2.

The required result follows by taking expectations. □

We summarize the above reductions with the following theorem

Theorem 4.15: *The expected completion time of model 1 is less than or equal to that of model 4.*

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. However, in order to conduct these protocols, the network should first execute a preparation protocol. This protocol is executed only once.

5.1 Preparation This protocol is executed during the set up phase (Section 2) before starting any point to point transmission. In Section 2 we described how the BFS tree is constructed, here we describe how additional information, the BFS parent, the BFS descendants and on which subtree each descendant belongs, is conveyed to each node.

The BFS protocol of Section 2 enables each node to know the ID of its BFS parent, and its depth (distance from the root). The descendant information can be found once 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., $O((n-1+D)\log\Delta) = O(n\log n)$ time. To record all this information each node must have sufficient storage to keep $O(n)$ IDs.

To save space (and time) we propose the following scheme [13]: After the BFS tree is completed, a depth-first-search (DFS) is conducted on the BFS tree. Henceforth, each node uses its DFS number as its address. Since the DFS numbers of the descendants of a node constitute a consecutive range, it suffices that each node remember the DFS number of each of its children and the maximum DFS number of all the descendants. Thus, each node v needs only $O(deg(v)\log n)$ bits of local memory.

Using a token, DFS can be implemented in $O(n)$ time: First the token conducts a DFS of the graph, each node sends the token to the largest neighbor not yet in the DFS tree and when all the neighbors are exhausted it is sent to the parent. Whenever a node sends the token it broadcasts its own ID together with the ID of its BFS-parent. Thus all the neighbors of a node know when the node joins the DFS-tree, and the token is not sent to nodes already in the tree (except when the DFS backtracks from a child to its parent). There are no conflicts, since only the node holding the token can transmit, also the entire traversal requires $2n-2$ time, since the token traverses once in each direction of each tree-edge.

After the first DFS is completed, each node knows the parents of all its neighbors, in particular it knows which of its neighbors are its BFS-children. Thus we can conduct a second DFS traversal this time on the BFS tree. This traversal also costs $O(n)$ and after it is completed each node knows the DFS-number of all its BFS-children and its maximum descendant. Note that we required that each node knows the IDs of all its neighbors only in order to conduct the first DFS-traversal.

5.2. The upward subprotocol This protocol is essentially identical to the collection protocol, except that messages do 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.

5.3. 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 designated 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 designated 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.

5.4. Performance analysis The setup phase requires $O(n + D\log\Delta)$ time after which passing k messages requires an average of $O((k+D)\log\Delta$ time. When $k\to\infty$ 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 $1-\varepsilon$. 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 (we allow an error of $\varepsilon=1/n^2$). At superphase t the root sends the tth 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 $\geq$ ½ that v receives 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 $\geq$ ½ 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 $\sum_{i=0}^{\infty} \frac{1}{n^i} = \frac{n}{n-1}$. Moreover, the time spent in each layer is $O(\log\Delta\log n)$, thus the effective rate in which messages leave the root is $O(\frac{n}{n-1}\log\Delta\log n) = O(\log\Delta\log n)$. 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 $3n^2$. 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 all the acknowledgements reach the root is $O((D\log n+n)\log\Delta)$. Let c be the implied constant in the above expression. If the root does not receive acknowledgements from all the nodes by $2c(D\log n+n)\log\Delta)$ time slots after it sent the n^2th message it resends the last n^2 messages. It can be shown that the probability that the n^2th message has to be resent is less than ½, thus this last correction increases 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 $1 \le id'_i \le n$ 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.

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 require $O(n + D\log\frac{N}{\varepsilon}\log\Delta)$ 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 postulated 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 and Moshe Sidi for helpful discussions and an anonymous referee for suggesting using DFS numbers to improve the memory requirements.

REFERENCES

[1] Alon N., Bar-Noy A., Linial N. and Peleg D., "A lower bound for radio broadcast", to appear in J. Computer and System Science, also in STOC 1989.

[2] Birk Y., "Concurrent communication among multi-transceiver stations over shared media" Ph.D. Thesis, Tech. report CSL-TR-87-321, Stanford University, Stanford, CA.

[3] Bar-Yehuda R., Goldreich O. and Itai A., "On the time-Complexity of Broadcast in Radio Networks: An Exponential Gap Between Determinism and Randomization", to appear in J. Computer and System Science, also in PODC 1987.

[4] Bar-Yehuda R., Goldreich O. and Itai A., Efficient Emulation of Single-Hop Radio Network with Collision Detection on Multi-Hop Radio Network with no Collision Detection, Distributed Computing, Vol. 5, 1991, pp. 67-71.

[5] Burke, P.J. "The Output of a Queuing System", *Operations Research,* 4, 699-704 (1956).

[6] Capetanakis, J., "Generalized TDMA: The multi-accessing tree protocol", *IEEE Trans. Commun.*, Vol. COM-27, 1479-1484, (1979).

[7] Chlamtac, I. and Kutten, S., "On Broadcasting in Radio Networks- Problem Analysis and Protocol Design", *IEEE Transactions on Communications,* December (1985), Vol COM-33, No. 12.

[8] Chlamtac, I. and Weinstein O., "The wave expansion approach to broadcasting in multi-hop radio networks", *INFOCOM*, 874-881, (April 1987).

[9] Digital-Intel-Xerox, "The Ethernet data link layer and physical layer specification 1.0" (Sept. 1980).

[1] Gallager, R., "A perspective on multiaccess channels", *IEEE Trans. on Inf. Theory*, Vol. IT-31 (1985), 124-142.

[11] Hofri, M., "A Feedback-less Distributed Broadcast Algorithm for Multihop Radio Networks with Time-varying Structure", *2nd ACM Intr. MCPR Workshop*, Rome (May 1987). Also available as TR-451, Computer Science Dept., Technion, Haifa, Israel (March 1987).

[12] Hsu and P.J. Burke, "Behavior of Tandem Buffers with Geometric Input and Markovian Output", *IEEE Trans. on Comm.*, COM-24, 359-361, (1976).

[13] Itai, A. and Rodeh, M., The Multi-Tree Approach to Reliability in Distributed Systems, Proceedings of the 25 Symposium on Foundations of Computer Science, 137-147, (Oct. 1984). Also in Information and Computation, vol. 79(1), 43-59, (Oct 1988).

[14] L. Kleinrock, *Queueing Systems, Volume 1: Theory,* John Wiley and Sons, (1975).

[15] Stoyan, D., *Comparison Method for Queues and other Stochastic Models,* J. Wiley & Sons, 1983.

[16] Weinstein O., "The wave expansion approach to broadcasting in multihop radio networks", M.Sc. Thesis, Computer Science Dept., Technion, Haifa, Israel, (1987).

[17] Willard D.E., "Log-logarithmic selection resolution protocols in a multiple access channel", SIAM J. on Comput. 15(2), 468-477, (1986).