Optimal Broadcast with Partial Knowledge

(Extended Abstract)

Baruch Awerbuch * Shay Kutten † Yishay Mansour ‡

David Peleg §

June 28, 1995

Abstract

This work concerns the problem of broadcasting a large message efficiently when each processor has partial prior knowledge about the contents of the broadcast message. The partial information held by the processors might be out of date or otherwise erroneous, and consequently, different processors may hold conflicting information.

The problem of Broadcast with Partial Knowledge (BPK) was introduced in the context of Topology Update - the task of updating network nodes about the network topology after topological changes. Awerbuch, Cidon, and Kutten gave a message optimal solution for BPK, yielding a message optimal Topology Update algorithm. However, the time complexity of both algorithms was not optimal. The time complexity was subsequently improved in two follow up papers, but the best known time complexity was still higher than optimal by at least a logarithmic factor.

In this paper we present a time-optimal, communication-optimal algorithm for BPK. The algorithm is randomized, and, similar to previous randomized algorithms, it does not require the additional knowledge assumptions essential for deterministic solutions. In addition to the theoretical interest in optimality, a logarithmic factor is often important in practice, especially when using the procedure as a component within a periodically activated Topology Update algorithm.

*Johns Hopkins University, Baltimore, MD 21218, and MIT Lab. for Computer Science (baruch@blaze.cs.jhu.edu). Supported by Air Force Contract TNDGAFOSR-86-0078, ARPA/Army contract DABT63-93-C-0038, ARO contract DAAL03-86-K-0171, NSF contract 9114440-CCR, DARPA contract N00014-J-92-1799, and a special grant from IBM.

†IBM T.J. Watson Research Center P.O. Box 704, Yorktown Heights, NY 10598 (kutten@watson.ibm.com).

‡Department of Computer Science, Tel Aviv University, Tel Aviv 69978, Israel (mansour@math.tau.ac.il).

§Department of Applied Mathematics and Computer Science, The Weizmann Institute, Rehovot 76100, Israel (peleg@wisdom.weizmann.ac.il). Supported in part by an Allon Fellowship, by a Walter and Elise Haas Career Development Award and by a Bantrell Fellowship.

1 Introduction

1.1 Motivation

Many tasks in distributed computing deal with concurrently maintaining the "view" of a common object in many separate sites of a distributed system. This object may be the topology of a communication network (in which case the view is a description of the underlying network graph), or certain resources held at the system sites (in which case the view is an inventory listing the resources held at each site), or even a general database. The objects considered here are dynamic in nature, and are subject to occasional changes (e.g., a link fails, a resource unit is consumed or released, a database record is modified). It is thus necessary to have an efficient mechanism for maintaining consistent and updated views of the object at the different sites.

One obvious algorithm for maintaining updated views of a distributed object is the *Full Broadcast* algorithm. This algorithm is based on initiating a broadcast of the entire view of the object whenever a change occurs. Due to the possibility of message pipelining, the time complexity of this algorithm is relatively low. On the other hand, this algorithm might be very wasteful in communication, since the object may be rather large.

Consequently, it is clear that a successful consistency maintenance strategy should strive to utilize the fact that the processors already have a correct picture of "most" of the object, and need to be informed of relatively few changes. Viewed from this angle, the problem can be thought of as having to broadcast the entire view of the object, while taking advantage of prior partial knowledge available to the processors of the system.

On the other extreme there is the *Incremental Update* strategy, in which only "necessary" information is transmitted. This strategy is at the heart of the algorithms suggested for handling the topology update problem ([ACK90, MRR80, SG89, BGJ+85]). Unfortunately, it is not obvious how to employ information pipelining with this method, as demonstrated in the sequel. This increases the time complexity.

The purpose of this work is to study the problem of updating a distributed database, under minimal assumptions. That is, we do not assume any initial coordination and allow only small amount of space. Under such conditions, we look for efficient solutions to the problem with respect to communication and time overheads. In this setting, it turns out that the main bottleneck of the database update problem can be characterized as a fairly simple "communication complexity" problem, called *Broadcast with Partial Knowledge.*

The problem of Broadcast with Partial Knowledge was introduced in the context of Topology Update - the task of updating network nodes about the network topology after topological changes. Awerbuch, Cidon, and Kutten gave a message optimal solution, thus yielding a message optimal Topology Update algorithm. However, the time complexity (of both the broadcast with partial knowledge, and the topology update) was not optimal. The time complexity was subsequently improved in two follow up papers, one providing a randomized algorithm (using fewer assumptions on knowledge) [ACK+91], and one giving a deterministic algorithm (using a stronger knowledge assumption, and with time complexity higher by a polylogarith-

mic factor)[AS91]. The time complexity of the randomized algorithm of [ACK+91] was still higher than optimal by at least a logarithmic factor. While this may not seem a large factor from the theoretical point of view, it is still interesting to find out that an optimal algorithm is possible. From the practical point of view, getting rid of a logarithmic factor is often important, especially in the context of the practically significant Topology Update problem.

In this paper we present a time optimal algorithm. The algorithm is randomized, and, similar to the previous randomized algorithm of [ACK+91], does not require the additional knowledge required by any possible deterministic algorithm. The message complexity of the algorithm is optimal too. The improvement in the time complexity is obtained by "simulating" the previous algorithm but in a less synchronized manner. This enables full pipelining, while in the previous algorithm there may have been times nodes could not transmit between "phases".

1.2 The model and the problem

The *Broadcast with Partial Knowledge* problem can be formulated as follows. Consider an asynchronous communication network, consisting of $n+1$ processors, $p_0, \ldots, p_n$, with each processor p_i has an m-bit *local input* w_i, and processor p_0 is distinguished as the *broadcaster*. In a correct solution to the problem all the processors write in their local output the value of the broadcaster's input, $w = w_0$.

This formulation of the problem can be interpreted as follows. The input w_i is stored at processor p_i and describes the local representation of the object at processor p_i. The correct description of the object is $w = w_0$, held by the broadcaster. The local descriptions w_i may differ from the correct one as a result of changes in the object. In particular, every two processors may have different descriptions due to different messages they got from the broadcaster in the past, as a result of message losses, topology changes and the asynchronous nature of the network. Our goal is to inform all the processors throughout the network about the correct view of the object w, and to use the processor's local inputs given to each processor in order to minimize the time and communication complexities.

In this paper, we provide a randomized solution for the hardest version of this problem, in which each processor only knows its own input, and has no information regarding inputs of other processors. A weaker version of the problem is based on making the rather strong "neighbor knowledge" assumption, namely, assuming that each processor correctly knows the inputs of all its neighbors, in addition to its own. This assumption is justified in [ACK90], where it is shown that neighbor-knowledge comes for free in context of database and topology update protocols. Even for this weaker problem, none of the previously known solutions are optimal *both* in communication and time. Following the randomized algorithm of [ACK+91], a deterministic algorithm was given for the weaker problem [AS91]. The complexity of that algorithm is larger than that of the randomized algorithm presented here by a polylogarithmic factor.

In order to quantify the possibility of exploiting local knowledge, we first introduce a new measure that captures the level of "information" of the knowledge held by each processor. Let the *discrepancy* δ_i of the input w_i held by processor p_i be the number of bits in which w_i, the local description at p_i, differs from the

broadcaster's input w, which is the correct description of the object. Define also the *total discrepancy* $\Delta = \sum_i \delta_i$, the *average discrepancy* $\delta_{av} = \Delta/n$, and the *maximum discrepancy* $\delta_{max} = \max_i\{\delta_i\}$.

Our goal is to study the relationships between these discrepancies and the complexity of broadcast algorithms, following the intuition that the complexity of broadcast protocols should be proportional to discrepancy of processors' inputs, i.e., if the views of most processors are "almost correct", then the overhead of the protocol should be small. We therefore express the communication and time complexity of our solution as a function of m, n and δ_{av}. The complexities are measured in the bit complexity model.

1.3 Basic solutions

The first obvious solution to the *Broadcast with Partial Knowledge* problem is the aforementioned *Full Broadcast* protocol, which is wasteful in communication, i.e. requires $\Omega(nm)$ bits. On the other hand it is rather fast, since the broadcast can be done in a pipelined fashion and thus can terminate in $O(n+m)$ time. Thus, one would like to improve on this algorithm with respect to communication complexity, aiming towards reducing this complexity to be close to the total discrepancy Δ, while maintaining near-optimal time complexity.

The *Incremental Update* strategy proposed in [ACK90] poses an alternative approach. It can be applied only under the strong assumption of "neighbor knowledge": each node is assumed to "know" the value of the database at its neighbor. The essence of this strategy is that a node with "correct" view transmits to its neighbor a "correction" list, which contains all the positions where neighbor's input is erroneous. When the neighbor received all the corrections, it can assume that the rest of its input is correct, and start using it for correcting its own neighbors who are further away from the source. Thus a "correction wave" propagates through the network from the source, till all nodes are corrected. Note that there is almost no pipelining possible in this algorithm as described above, since a node can start transmitting only after it done with the receiving. Even in the simple case of a path network the protocol may require $\Omega(\Delta)$ time. As mentioned above, in the follow-up paper [AS91] the complexity of this strategy (for the "neighbor knowledge" variant of the problem) was improved significantly (although still not matching the lower bound), using a very sophisticated partitioning of the information, and a recursive implementation.

1.4 Our results

In this paper, we provide an efficient randomized solution to the Broadcast with Partial Knowledge problem. It has success probability at least $1-\epsilon$, and uses $O(\Delta \log m + n \log \frac{n}{\epsilon})$ communication and $O(n+m)$ time, where ϵ is a parameter to the algorithm. The algorithm can be easily adapted to be executed on trees, rather than on a line (which is the "worst-case" tree) as described here. Thus in an arbitrary topology network, performing the algorithm on a shortest-path tree yields the same message complexity, with the time complexity being $O(D+m)$, where D

Algorithm	Communication	Time	
Full broadcast (folklore)	nm	$n+m$	
Incr. Update [ACK90]	$n+\Delta \log m$	$n+\Delta \log m$	NK
[ACK+91]	$\Delta \log m + n \log \frac{n}{\epsilon}$	$n \log \delta_{av} m + \log \frac{1}{\epsilon}\}$	
[AS91]	$(\delta_{av}+1)n \log m$	$(n+m)\log^3 m$	NK
Our algorithm	$\Delta \log m + n \log \frac{n}{\epsilon}$	$n + \log \frac{n}{\epsilon} + \min\{m, \Delta \log m\}$	
Lower bounds [ACK+91]	$n + n\delta_{max} \log(m/\delta_{max})$	$n + \delta_{max} \log(m/\delta_{max})$	

Figure 1: Comparison of protocols and lower bounds. "NK" stands for the neighbor knowledge assumption.

is the network diameter. Note that in all cases, we allow the inputs stored at the various processors to differ in arbitrary ways, subject to the discrepancy constraints.

Our upper bounds are derived using linear codes. Such codes were used before in constructing distributed algorithms for solving various problems. In particular they are used in a similar way by [ACK+91]. Metzner [Met84] uses Reed-Solomon and random codes to achieve efficient retransmission protocols in a complete network. Ben-Or, Goldwasser and Wigderson [BGW88] use BCH codes to guarantee privacy in a malicious environment. Rabin [Rab89] uses codes to achieve a reliable fault-tolerant routing with a low overhead.

In [ACK+91] it was shown that when the average discrepancy is δ_{max}, the communication complexity is at least $\Omega(\Delta \log(\frac{m}{\delta_{max}}))$ and the time complexity is at least $\Omega(n+\delta_{max} \log(\frac{m}{\delta_{max}}))$. It is also argued there that in the case that no information is known about the discrepancies, any deterministic protocol would send $\Omega(nm)$ bits, even if there are *no* discrepancies at all.

1.5 Applications

One application of our work is to the classical network problem of Topology Update, which is at the heart of many practical network protocols [MRR80, BGJ+85, ACG+90]. In Topology Update, initially, each processor is aware of the status of its adjacent links, i.e., whether each link is up or down, but may not be aware of the status of other links. The purpose of the protocol is to supply each processor with this global link status information.

The topology update algorithm of [ACK90] is based on the *Incremental Update* strategy. The possibility of recurring network partitions and reconnections significantly complicates implementation of this strategy. Nevertheless, the resulting broadcast procedure is efficient in terms of communication (although not in time), and leads to essentially communication-optimal topology update protocols [ACK90].

A consequence of [ACK90] that is most significant for our purposes is the observation that it is possible to relate the complexities of the problem of Broadcasting with Partial Knowledge to those of Topology Update, effectively reducing the latter problem to the former. Namely, given *any* solution for the Broadcast with Partial Knowledge problem, one can construct a topology update protocol with a similar overheads in both communication and time.

Reference	Amortized Commun.	Quiescence Time
Full broadcast ([AAG87])	VE	$V+E$
Incr. Update [ACK90]	$V \log E$	$V^2 \cdot \log E$
Our upper bound	$V \log E$	$E + V \log V$
Lower bounds	$V \log E$	E

Figure 2: Applications of Partial Knowledge Broadcast protocols to topology update.

Figure 2 summarizes the complexities of protocols to the topology update task obtained by applying various Broadcast with Partial Knowledge algorithms (with V, E denoting the number of vertices and edges in the network, respectively).

It is worth pointing out that our complexity results are presented in the bit complexity model, whereas the results in [ACK90] are presented in the message complexity model which charges only one complexity units for a message of size $O(\log n)$ bits.

Our algorithm may also be applicable for dealing with the issue of self stabilization [Dij74, ASY90, KP90, APV91]. The self-stabilization approach is directed at dealing with intermittent faults that may change the memory contents of nodes, and cause inconsistency between the local states of nodes. Dijkstra's example [Dij74] is that of a token passing system, where it is required that exactly one of the nodes holds a *token* at any given time. The faults may cause an illegal situation in which no node holds a token (each "assumes" that some other has it) or alternatively, that several nodes hold a token. Overcoming such faults requires the nodes to continuously check the states of their neighbors, and possibly to correct them when necessary. It is conceivable that the faults cause only partial changes in memory, so our algorithm can be used to mend the situation. Note that in this context, it is essential that we do not make the "neighbor knowledge" assumption.

1.6 Organization of the paper

The rest of the paper is structured as follows. In section 2 we review for later use some necessary material concerning universal hash functions and coding theory. This background is mostly the same as in [ACK+91]. In Section 3, we first describe the algorithms of [ACK+91], that motivate and help in the proof of the new result, which is algorithm BPART in Subsection 3.4. The full proof of the new result is given in [ACK+95], but we do point out similarities between the proofs and parts of the proof of the previous algorithms. This is possible since the new algorithm is basically a simulation of the previous algorithms, but with enhanced pipelining. It is hoped that the new pipelining technique can be used to help pipeline other phase based algorithms.

2 Preliminaries

2.1 Universal hash functions

Universal hash functions have found many interesting applications since their introduction by Carter and Wegman [WC79]. A family of functions $\mathcal{F} = \{h : A \to B\}$ is called a *universal hash function* if for any $a_1 \neq a_2 \in A$ and $b_1, b_2 \in B$ the following holds:

$$Prob[h(a_1) = b_1 \text{ and } h(a_2) = b_2] = \frac{1}{|B|^2}$$

where the probability is taken over the possible choices of h, which is randomly and uniformly chosen from $\mathcal{F}$.

There are many families of simple universal hash functions. One example can be constructed as follows. Let p be a prime and let $B = Z_p$. (Note $|B| = p$.) Then

$$H = \{h_{\alpha,\beta}(x) = (\alpha x + \beta) \bmod p \mid \alpha, \beta \in Z_p\}$$

is a family of universal hash functions.

In the above example the encoding of a hash function requires only two elements from Z_p, and also p, therefore we can describe such a hash function using only $O(\log |B|)$ bits. (Note that the encoding of h does not depend on A.) Later, when using a universal hash function, it is assumed that it can be represented with $O(\log |B|)$ bits.

Another way to view the parameters is the following. We are interested in a family of universal hash functions $\mathcal{F}_\epsilon$, that has the following property: given any two distinct elements, the probability that a random hash function $h \in \mathcal{F}_\epsilon$ maps them to the same point, is bounded by ϵ. From the properties of the universal hash function this occurs with probability $1/|B|$. Therefore, choosing $\epsilon = 1/|B|$, we conclude that there is a family of hash functions $\mathcal{F}_\epsilon$ whose encoding size is $O(\log \frac{1}{\epsilon})$.

2.2 Information theoretic background

The tools developed later on are based on some basic results from coding theory. A code $C_{m,d} : \{0,1\}^m \mapsto \{0,1\}^{m+r}$ is a mapping that transforms an input word $w \in \{0,1\}^m$ into a codeword $C_{m,d}(w) = \hat{w} \in \{0,1\}^{m+r}$. The codes considered in this paper are standard "check-bit" codes, namely, the resulting codeword $\hat{w}$ is assumed to be of the form $\hat{w} = w||\rho$, where $\rho \in \{0,1\}^r$ is a "trail" of r check bits concatenated to the input word w, called the *syndrom*. Denote the check bit syndrom that the code $C_{m,d}$ attaches to a word w by $C^*_{m,d}(w)$. The lengths of the entire codeword and the check bit syndrom are denoted in the sequel by $|C_{m,d}(w)|$ and $|C^*_{m,d}(w)|$, respectively.

A code $C_{m,d}$ is said to be *d-correcting* if the original word w can be correctly decoded from any word z that differs from the codeword $C_{m,d}(w)$ in no more than d places.

The following theorem states the properties possessed by the code necessary for our purposes.

Theorem 2.1 ([ACK+91]) For any m and $d \leq m/3$, there exists a check-bit code $C_{m,d}$ with the following properties:

1. The check bit syndrom is of length $|C^*_{m,d}(w)| = O(d \log m)$.
2. The code $C_{m,d}$ is d-correcting.
3. The encoding and decoding operations ($C_{m,d}$ and $C^{-1}_{m,d}$, respectively) require time polynomial in m and d.

In order to prove the theorem, [ACK+91] modified BCH codes slightly. All codes $C_{m,d}$ referred to later on in the paper are meant to be check-bit codes that satisfy the properties in Theorem 2.1. The subscripts m, d are omitted whenever m and d are clear from the context.

3 Upper bounds

We develop our solution in a modular fashion via a number of steps. The first three steps follow [ACK+91]. The first step is a simple algorithm named MAXIMUM, presented in Subsection 3.1, which is based on the assumption that the maximum discrepancy δ_{max} is known to the broadcaster. Next, Subsection 3.2 presents the algorithm AVERAGE, which assumes only knowledge of the average discrepancy. Then, in Subsection 3.3 it is shown that the assumptions about knowledge of the discrepancy can be eliminated at the cost of increasing the time complexity by a factor of $\log \delta_{av}$. Finally, in Subsection 3.4 we present our new algorithm, based on condensing the previous algorithm, so that its time complexity becomes optimal again.

3.1 Algorithm MAXIMUM

This section handles broadcast in the case where the maximum discrepancy δ_{max} is known, and presents a straightforward broadcasting algorithm MAXIMUM, which assumes that the broadcaster "knows" δ_{max}. The algorithm requires $O(n\delta_{max} \log m)$ communication and $O(n + \delta_{max} \log m)$ time.

We should note that this algorithm is not efficient, since the maximum discrepancy can be very far from the average discrepancy. This algorithm is presented, in order to be used in the next section as a subroutine.

For simplicity, it is assumed that the network is a simple path, namely, the $n+1$ processors $p_0, \ldots, p_n$ are arranged on a line, with a bidirectional link connecting processor p_i to processor p_{i+1}, for every $0 \leq i < n$. Note that this does not restrict generality in any way, since the path is the worst topology for broadcast, and moreover, there exists an easy transformation from every other network to a path network by using a depth-first tour ([Eve79]).

Algorithm MAXIMUM works as follows: The *broadcaster* encodes the broadcast message w using the code $C = C_{m,\delta_{max}}$. (Note that this code C is fixed and known to all other processors.) The broadcaster broadcasts only the check bit syndrom $C^*(w)$. The broadcasting proceeds in full pipelining. I.e., each processor p_i for $i < n$ that receives the first bit of $C^*(w)$ immediately forwards it to processor p_{i+1}, without waiting for the entire value of $C^*(w)$.

Once a processor p_i has received the complete message $\rho = C^*(w)$, it concatenates it to its own input w_i, thus obtaining a complete (but possibly corrupted)

codeword $\hat{w}_i = w_i || \rho$ and decodes this codeword by computing $o_i = C^{-1}(\hat{w}_i)$, which is taken to be the output.

Lemma 3.1 ([ACK+91]) If the input w_i of processor p_i is different from w in at most δ_{max} places, then $o_i = w$.

Lemma 3.2 ([ACK+91]) The time Complexity of Algorithm MAXIMUM is $n + O(\delta_{max} \cdot \log m)$, and its communication complexity is $O(n \cdot \delta_{max} \log m)$.

We complete the description by noting that both the time and communication complexities can be improved for large δ_{max}. Specifically, if $\delta_{max} \log m > m$, then a full broadcast of the information is more efficient (namely, send w to all the processors). Therefore we have

Theorem 3.3 ([ACK+91]) Given the value of δ_{max}, there is a deterministic algorithm for performing broadcast with partial information, that requires $n + O(\min\{m, \delta_{max} \cdot \log m\})$ time and has communication complexity $O(n \cdot \min\{m, \delta_{max} \cdot \log m\})$.

A similar result holds when the broadcaster knows only an upper bound d on the discrepancies, where the same complexities hold except with d replacing δ_{max}. When the upper bound is "accurate", namely $d = O(\delta_{max})$, the complexities remain the same.

Note that the communication complexity of this algorithm is not good when there are differences between the discrepancies of nodes. For example, consider the case that one one has a high discrepancy. This forces the algorithm to send a long check bit syndrom also to the other nodes on the way, although they have low discrepancies. The algorithm of the next subsection manages to fix this problem, even though it relies on a weaker assumption.

3.2 Algorithm AVERAGE

In this section we replace the assumption of known δ_{max} with the assumption that the average discrepancy δ_{av} is known. Note that no assumptions are made about how the discrepancies are distributed. In particular, it may be that some processors have large discrepancies while others have the correct value. For simplicity of notation, we assume throughout the section that $\delta_{av} \geq 1$, or $\Delta \geq n$.

The broadcast algorithm AVERAGE presented in this section is randomized, i.e., it guarantees the correctness of the output of each processor with high probability. The communication complexity of Algorithm AVERAGE depends linearly on the average discrepancy δ_{av}, while its time complexity is still linear in m. Both complexities apply to the worst case scenario.

We begin with a high level description of Algorithm AVERAGE. The algorithm works in phases, and invokes Algorithm MAXIMUM of Section 3.1 at each phase. At every phase of the execution, each processor can be in one of two states, denoted $\mathcal{K}$ and $\mathcal{R}$. Initially, only the broadcaster is in state $\mathcal{K}$, while the other processors are in state $\mathcal{R}$. Intuitively, a processor p_i switches from state $\mathcal{R}$ to state $\mathcal{K}$ when it concludes that his current guess for w is equal to the "real" broadcast word w.

The phases are designed to handle processors with increasingly larger discrepancies. More specifically, let us classify the processors into classes $C_1, \ldots, C_\mu$, $\mu = \left\lceil \log(\frac{m}{\delta_{av} \log m}) \right\rceil$, where the class C_l contains all processors p_i whose discrepancy δ_i falls in the range $2^{l-1}\delta_{av} < \delta_i \leq \min\{m, 2^l \delta_{av}\}$ for $2 \leq l \leq \mu - 1$, $\delta_i \leq 2\delta_{av}$ for $l = 1$, and the rest in C_μ (i.e., $\delta_i \geq \frac{m}{\log m}$). Then each phase $l \geq 0$ is responsible for informing the processors in class C_l. This is done by letting the processors in state $\mathcal{K}$ broadcast to the other processors.

Note that the $\mathcal{K}$ and $\mathcal{R}$ states reflect, in a sense, only the processors' "state of mind", and not necessarily the true situation. It might happen that a processor switches prematurely to state $\mathcal{K}$, erroneously believing it holds the true value of the input w. Such an error might subsequently propagate to neighboring processors as well. Our analysis will show that this happens only with low probability.

By a simple counting argument, the fraction of processors whose discrepancy satisfies $\delta_i \geq k\delta_{av}$ is bounded above by $\frac{1}{k}$, for every $k \geq 1$. The first phase attempts to correct the inputs of processors from C_1, while in general, the l-th phase attempts to correct the processors in C_l. By the previous argument, at least half of the processors are in C_1, and furthermore, $\sum_{j=l}^{\mu} |C_j| \leq \frac{n}{2^l}$. Assuming that all the processor that shifted from $\mathcal{R}$ to $\mathcal{K}$ had the correct value, then after the l-th phase, at most $\frac{n}{2^l}$ processors are in state $\mathcal{R}$.

Let us now describe the structure of a phase l in more detail. At the beginning of phase l, the current states of the processors induces a conceptual partition of the line network into consecutive intervals $I_1, \ldots, I_t$, with each interval $I = (p_i, p_{i+1}, \ldots)$ containing one or more processors, such that the first processor p_i is in state $\mathcal{K}$, and the rest of the processors (if any) are in state $\mathcal{R}$.

The algorithm maintains the property that each processor knows its state, as well as the state of its two neighbors, hence each processor knows its relative role in its interval, as either a "head" of the interval, an intermediate processor, or a "tail" (i.e., the last processor of the interval).

Suppose that processor p_i is in state $\mathcal{K}$ at the beginning of phase $l < \mu$ and is the "head" of some interval I. If p_{i+1} is also in state $\mathcal{K}$, then the interval I contains only p_i, and thus p_i has finished its part in the algorithm. Otherwise, interval I contains at least one processor in state $\mathcal{R}$. In this case, processor p_i is assigned the role of the *broadcaster* with respect to its interval in phase l. More specifically, it needs to inform its value to all processors of class C_l in its interval I. Hopefully, this results in the further partition of interval I into subintervals for the next phase.

Processor p_i performs this task by using Algorithm MAXIMUM of Section 3.1, with parameter $d_l = 2^l \delta_{av}$. To be more specific, if p_i's interval I contains other processors (i.e., processor p_{i+1} is in state $\mathcal{R}$) then p_i computes $C^*_{m,d_l}(o_i)$ and sends it to p_{i+1}. (In case $l = \mu$, processor p_i sends o_i.) As we shall see, with high probability $o_i = w$ for any processor i that is in state $\mathcal{K}$. Therefore, later in this informal description we substitute $C^*_{m,d_l}(w)$ for $C^*_{m,d_l}(o_i)$. Consider any intermediate processor p_j (in state $\mathcal{R}$) in interval I that receives a message simply forwards the message (using pipelining). The tail processor of the interval (i.e., the one whose successor is in state $\mathcal{K}$) does nothing.

It remains to explain when a processor decides to change its state from $\mathcal{R}$ to $\mathcal{K}$. This task requires an *initialization* phase, in which the broadcaster chooses a

random universal hash function $h \in \mathcal{F}_{\epsilon/n\mu}$ and sends both the description of h and the hashed value of the broadcast message, i.e., the pair

$$\mathcal{H}(w) = \langle h, h(w) \rangle,$$

to all processors. Since the description of h requires $O(\log \frac{n\mu}{\epsilon})$ bits, the size of the message is $O(\log \frac{n\mu}{\epsilon}) = O(\log \log m + \log \frac{n}{\epsilon})$. The pair $\mathcal{H}(w)$ will later serve each processor to test whether its new computed value of w is correct.

Specifically, as said above, in phase $l < \mu$ each processor p_j in state $\mathcal{R}$ receives $\rho_l = C^*_{m,d_l}(w)$. It concatenates it to w_j and computes

$$g_j(l) = C^{-1}_{m,d_l}(w_j C^*_{m,d_l}(w)),$$

which is its "guess" for w. It then tests whether $h(g_j(l)) = h(w)$.

In case of equality, the processor deduces that its current guess is correct. It then changes its state from $\mathcal{R}$ to $\mathcal{K}$, and sets its output to be $o_j = g_j(l)$. At the last phase, $l = \mu$, when a processor p_j receives value o_i, from some processor p_i, then p_j sets $o_j = o_i$. The algorithm ends after phase μ.

Lemma 3.4 ([ACK+91]) The probability that some processor produces an incorrect output is bounded above by ϵ.

Now, we analyze the communication and time complexity of the protocol. We first show that if no node mistakenly outputs an incorrect value, then both time and communication complexities are small.

Lemma 3.5 ([ACK+91]) Assuming that no processor outputs an incorrect value, the time complexity of Algorithm AVERAGE is $O(n + m + \log \frac{1}{\epsilon})$, and its communication complexity is $O(\Delta \log m + n \log \frac{n}{\epsilon})$.

Note that ϵ can always be chosen so as to make the failure probability polynomially small in m, without degrading the time or communication complexities of the algorithm. Consequently we have

Theorem 3.6 Given an average discrepancy δ_{av} and $0 < \epsilon < 1$, Algorithm AVERAGE solves the broadcast with partial knowledge problem correctly with probability $1 - \epsilon$. In the case that the solution is correct, the time complexity is $O(n + m + \log \frac{1}{\epsilon})$ and the communication complexity is $O(\delta_{av} \cdot n \cdot \log m + n \log \frac{n}{\epsilon})$ bits.

In case algorithm AVERAGE fails, and the output is incorrect, we can guarantee only trivial bounds on the time and communication complexities of the algorithm. These bounds are derived from bounding the number of phases by $\log m$, and the number of bits in a message by m. This gives a worst case bounds of $O((n+m) \log m)$ time and $O(nm)$ communication. However, ϵ can be selected so as to equate the expected complexity (over all executions) with the high probability complexity (i.e. over the executions that have a correct output) . Consequently we have

Corollary 3.7 Algorithm AVERAGE has expected time complexity $O(n + m)$ and expected communication complexity of $O(\Delta \log m + n \log nm)$ bits.

3.3 Unknown discrepancy

In the case that δ_{av} is not known in advance, we can solve the problem by initiating Algorithm AVERAGE with a guessed average discrepancy $\hat{\delta}_{av} = 1$. Call this Algorithm UNKNOWN. In such a case, in the $\log \delta_{av}$ first phases of Algorithm AVERAGE, it may happen that no processor changes to $\mathcal{K}$. The communication complexity essentially remain the same, since the additional $O(\delta_{av} \cdot n) = O(\Delta)$ bits are absorbed in the previous bound. However, the time complexity does increase in this case by an additive factor of $O(n \log \delta_{av})$.

Theorem 3.8 ([ACK+91]) Given $0 < \epsilon < 1$, Algorithm UNKNOWN solves the broadcast with partial knowledge correctly with probability $1 - \epsilon$. In the case that the solution is correct, the time complexity is $O(n \log \delta_{av} + m + \log \frac{1}{\epsilon})$ and the communication complexity is $O(\Delta \log m + n \log \frac{n}{\epsilon})$ bits.

In a similar way to the previous subsection, we can bound the expected complexities by choosing ϵ appropriately.

Corollary 3.9 ([ACK+91]) Algorithm UNKNOWN has expected time complexity $O(n \log \delta_{av} + m)$ and expected communication complexity of $O(\Delta \log m + n \log nm)$ bits.

3.4 Optimal Time Complexity

Intuitively, the main reason that the time in the algorithm of the previous subsection is not optimal is the fact that there can be "time spaces" between the phases. In this section we describe a simulation of Algorithm UNKNOWN in which the information transmitted is fully pipelined. For the sake of clarity we describe the whole algorithm below.

Algorithm BPART

We assume that $\Delta \log m < mn$. If this assumption does not hold, then the algorithm might fail, at which time full broadcast can be engaged, since the maximum complexity must be paid in any case.

The algorithm requires an initialization phase similar to the one of Algorithm AVERAGE, in which a random universal hash function $h \in \mathcal{F}_{\epsilon/n\mu}$ is chosen and the pair $\mathcal{H}(w)$ is sent to all processors.

The main part of the algorithm proceed as follows. Set $\mu = \log m - \log \log m$. The source transmits the encoding $w(i) = \mathcal{C}_{m,2^i}(w)$ of its vector w in all codes $\mathcal{C}^*_{m,2^i}$ of all levels $0 \leq i \leq \mu$, one after another. Hence the sequence transmitted by the source is of the form

$$\xi(w) = \langle w(0), w(1), \ldots, w(\mu), w \rangle.$$

Essentially, this information is to be forwarded along the line to all processors. Observe that a naive implementation of this would require communication complexity nm and time $m + n$. The communication complexity is minimized by stopping the flood of bits into a node p_j once this node is able to correctly decode the entire source's vector ξ (with high probability), i.e., to produce an output o_j such that

$h(w) = h(o_j)$. This is done in the same way as in Algorithm UNKNOWN, as described below.

During the main part of the algorithm, each processor p_j on the line does the following. It initially enters state $\mathcal{R}$, intuitively signifying the fact that its data may be outdated. It then receives the sequence $\xi(o_{j-1})$ from its predecessor p_{j-1}, constructs its own output o_j and sequence $\xi(o_j)$, and sends this sequence to its successor. The incoming sequence is received and processed entry by entry, as long as p_j is in state $\mathcal{R}$.

Each arriving entry $o_{j-1}(i) = C^*_{m,2^i}(o_{j-1})$, $0 \leq i \leq \mu$, is processed as follows. First, the entry is stored as $o_j(i)$, and forwarded to p_{j+1}. In addition, p_j concatenates $o_{j-1}(i)$ to w_j, and computes its "guess" for w,

$$g_j(i) = C^{-1}_{m,2^i}(w_j \cdot C^*_{m,2^i}(o_{j-1})),$$

It then tests whether $h(g_j(i)) = h(w)$.

In case of equality, the processor does the following. First, it changes its state from $\mathcal{R}$ to $\mathcal{K}$, and informs its predecessor p_{j-1} to stop sending the rest of the sequence $\xi(o_{j-1})$. Next, it sets its output to be $o_j := g_j(i)$, and computes the rest of the sequence $\xi(o_j)$ locally (in order to be able to continue sending it to its successor on the line).

Note that p_j may fail in its tests in all stages $i \leq \mu$. In this case, it will receive the entire sequence $\xi(o_{j-1})$ from its predecessor, including o_{j-1} in its last entry, and adopt this value for o_j.

Let us now bound the probability of failure.

Lemma 3.10 The probability that some processor produces an incorrect output is bounded above by ϵ.

Finally, we analyze the communication and time complexity of the protocol.

Theorem 3.11 Assuming that no processor outputs an incorrect value, the communication complexity of Algorithm BPART is $O(\Delta \log m + n \log \frac{n}{\epsilon})$.

Theorem 3.12 Assuming that no processor outputs an incorrect value, the time complexity of Algorithm BPART is $O(n + \log \frac{n}{\epsilon} + \min\{m, \Delta \log m\})$.

As before, we bound the expected complexities by choosing ϵ appropriately.

Corollary 3.13 Algorithm BPART has expected time complexity $O(n + m)$ and expected communication complexity $O(\Delta \log m + n \log nm)$.

References

[AAG87] Yehuda Afek, Baruch Awerbuch, and Eli Gafni. Applying static network protocols to dynamic networks. In 28^{th} *Annual Symposium on Foundations of Computer Science*, October 1987.

[ACG+90] Baruch Awerbuch, Israel Cidon, Inder Gopal, Marc Kaplan, and Shay Kutten. Distributed control for paris. In *Proc. 9th ACM Symp. on Principles of Distributed Computing*, 1990. To appear.

[ACK90] Baruch Awerbuch, Israel Cidon, and Shay Kutten. Optimal maintenance of replicated information. In *Proc. 31st IEEE Symp. on Foundations of Computer Science*. Comp. Soc. of the IEEE, IEEE, 1990.

[ACK+91] Baruch Awerbuch, Israel Cidon, Shay Kutten, Yishay Mansour, and David Peleg. Broadcast with partial knowledge. In *Proc. 10th ACM Symp. on Principles of Distributed Computing*, 1991.

[ACK+95] Baruch Awerbuch, Israel Cidon, Shay Kutten, Yishay Mansour, and David Peleg. Optimal broadcast with partial knowledge. Manuscript, available on request. 1995.

[APV91] Baruch Awerbuch, Boaz Patt-Shamir, and George Varghese. Self-stabilization by local checking and correction. In *Proc. 32nd IEEE Symp. on Foundations of Computer Science*, pages 268–277, October 1991.

[AS91] Baruch Awerbuch and Leonard J. Schulman. The maintenance of common data in a distributed system. In *Proc. 32nd IEEE Symp. on Foundations of Computer Science*, October 1991.

[ASY90] Y. Afek, S.Kutten, and M. Yung. Memory-efficient self-stabilization on general networks. In *Proc. 4th Workshop on Distributed Algorithms*, Italy, September 1990.

[BGJ+85] A. E. Baratz, J. P. Gray, P. E. Green Jr., J. M. Jaffe, and D.P. Pozefski. Sna networks of small systems. *IEEE Journal on Selected Areas in Communications*, SAC-3(3):416–426, May 1985.

[BGW88] Michael Ben-Or, Shafi Goldwasser, and Avi Wigderson. Completeness theorem for non-cryptographic fault tolerant distributed computing. In *Proc. 20th ACM Symp. on Theory of Computing*, May 1988.

[Dij74] Edsger W. Dijkstra. Self stabilizing systems in spite of distributed control. *Commun. of the ACM*, 17:643–644, 1974.

[Eve79] Shimon Even. *Graph Algorithms*. Computer Science Press, 1979.

[KP90] Shmuel Katz and Kenneth Perry. Self-stabilizing extensions for message-passing systems. In *Proc. 10th ACM Symp. on Principles of Distributed Computing*, Quebec City, Canada, August 1990.

[Met84] J. J. Metzner. An improved broadcast retransmission protocol. *IEEE Trans. on Communications*, COM-32(6):679–683, June 1984.

[MRR80] I. McQuillan, I. Richer, and E.C. Rosen. The new routing algorithm for the arpanet. *IEEE Trans. on Commun.*, COM-28, May 1980.

[Rab89] M. Rabin. efficient dispersal of information for security, load balancing, and fault tolerance. *J. of the ACM*, 36(3):335–348, 1989.

[SG89] John M. Spinelli and Robert G. Gallager. Broadcasting topology information in computer networks. *IEEE Trans. on Commun.*, May 1989. to appear.

[Tiw84] P. Tiwari. Lower bounds on communication complexity in distributed computer networks. In 25^{th} *Annual Symposium on Foundations of Computer Science, Singer Island, Florida*, pages 109–117, 1984.

[WC79] M.N. Wegman and J.L. Carter. Universal classes of hash functions. *Journal of Computer and System Sciences*, 18:143–154, 1979.

[Yao79] Andy Yao. Some complexity questions related to distributed computing. In *Proceedings of the* 11^{th} *Annual ACM Symposium on Theory of Computing, Atlanta, Georgia*, pages 209–213. ACM SIGACT, ACM, April 1979.