Science of Computer Programming 6 (1986) 1-9
North-Holland



# DERIVATION OF A DISTRIBUTED ALGORITHM FOR FINDING PATHS IN DIRECTED NETWORKS*

Robert McCURLEY and Fred B. SCHNEIDER
*Department of Computer Science, Cornell University, Ithaca, NY 14853, U.S.A.*

Communicated by L. Lamport
Received December 1983
Revised August 1984

**Abstract.** A distributed algorithm is developed that can be used to compute the topology of a network, given that each site starts with information about sites it is adjacent to, the network is strongly connected, and communication channels are unidirectional. The program is derived using assertional reasoning.

## 1. Introduction

Computer-communication networks implemented by radio channels present some interesting problems. Due to local terrain and antenna placement, sites might be able to send messages directly to sites from which they cannot receive messages directly. We call such a network a *directed network.* If there is a directed path between every pair of sites in a directed network, then every site can communicate with all other sites. To do this, sites must be aware of the topology of the network so that messages can be forwarded over appropriate routes.

An algorithm to compute and disseminate the topology of a directed network is derived in this paper. The algorithm is actually more general, since it can be used to disseminate the union of the information known to each site to all sites in the network. While the algorithm is itself interesting, our major concern in this paper is with its derivation. Techniques usually associated with developing sequential programs [1, 4] are used in developing our distributed program.

Section 2 gives some definitions pertaining to directed networks. In Section 3, the algorithm is derived. Section 4 contains some conclusions and mentions related work.

* This work is supported in part by NSF Grants MCS-8103605 and DCR-8320274. Schneider is also supported by an IBM Faculty Development Award.

## 2. Directed networks

A directed network is modeled by a set of sites $P$ and a set of *links* $L \subseteq P \times P$. $L$ models the communication structure of the network; link $(i, j) \in L$ iff site $j$ can receive directly messages sent by site $i$. Communication between all pairs of sites is possible only if the graph $(P, L)$ is strongly connected. In the following, we consider only strongly connected directed networks.

Each link in a directed network is assumed to implement a *virtual circuit* [7] with the following properties:

VC1: Every message sent is delivered.
VC2: Messages on a virtual circuit are delivered in the order sent.

A virtual circuit behaves like a FIFO queue—messages received are removed from the front of the queue and messages sent are appended to the rear. Implementation of a communications service satisfying VC1 and VC2 is presumed to be done by low-level software and will not be considered here. (An acknowledgment-retry protocol cannot be used because a channel for the acknowledgment message may not exist, but other techniques, such as repeated transmission of messages, or use of error-correcting codes, could be used.)

In a directed network, $i$ is a *predecessor* of $j$ and $j$ a *successor* of $i$ if $(i, j) \in L$. The set of predecessors of a site $i$ is denoted[1] by $pred_i$; the set of successors by $succ_i$.

For any site $i$ we assume the

**Local Topology Assumption.** $pred_i$ is the only information about $L$ that is initially available to site $i$.

This assumption reflects the fact that, in a network implemented by radio channels, a site initially knows about only those sites from which it can directly receive messages.

Communication between a site and its successors takes place using a **broadcast** statement, which corresponds to a radio broadcast. To send a message $m$ to all successors, a site $i$ executes the statement

**broadcast** $m$.

The effect of this is to append $m$ to the end of the message queue associated with each link $(i, j)$, $j \in succ_i$, after some unpredictable but finite delay.

To receive a message from a particular predecessor site $k$, $i$ executes a **receive** statement

**receive** $x$ **from** $k$.

[1] Assertions, values and variables associated with a site are subscripted with the name of that site.

This removes the first message from the queue associated with link $(k, i)$ and assigns it to $x$; if the queue is empty, site $i$ is delayed until a message from site $k$ has been delivered.

## 3. An algorithm for directed networks

We begin by defining the following functions on sites:

$|j, k| \equiv$ the length of the shortest directed path from site $j$ to site $k$,

$$diam(k) \equiv \max_{j \in P} |j, k|.$$

Because the network is strongly connected, $|j, k|$ and $diam(k)$ are total.

Now, consider a directed network in which a set $W_i$ is stored at each site $i$, where[2]

**W-Assumption.** $(\forall i: i \in P: W_i \neq \emptyset) \wedge (\forall i, j: i, j \in P: i = j \vee W_i \cap W_j = \emptyset)$.

For each site $i$, define[3]

$$Q_i^t \equiv (\bigcup j: |j, i| = t: W_j) \quad \text{for } 0 \leq t.$$

Thus, $Q_i^t$ contains those values that appear in set $W_j$ at each site $j$ that is connected to $i$ by a shortest path of length $t$.

We now derive an algorithm that establishes $S_i = (\bigcup j: j \in P: W_j)$ at each site $i$, or equivalently

$$R_i: \quad S_i = (\bigcup j: 0 \leq j \leq diam(i): Q_i^j).$$

Such an algorithm can be used to compute and distribute the topology of a directed network by using $W_i = pred_i \times \{i\}$.

### *3.1. The loop at site i*

Site $i$ uses a loop to establish $R_i$. The loop is developed from a loop invariant, which is obtained by generalizing $R_i$. $R_i$ can be weakened by replacing the constant $diam(i)$ by an integer variable $c_i$ to obtain the loop invariant:

$$P0_i: \quad S_i = (\bigcup j: 0 \leq j \leq c_i: Q_i^j) \wedge 0 \leq c_i.$$

Replacing a constant by a variable is one of the standard techniques described in [4] for obtaining a loop invariant from a result assertion. $P0_i$ asserts that $S_i$ contains values in sets $W_k$ for all sites $k$ connected to $i$ by a directed path of length at most $c_i$.

[2] It is not difficult to make arbitrary sets satisfy the W-Assumption: The set $W_k \times \{k\}$ is used in place of $W_k$, for all sites $k$.

[3] It is convenient in assertions to denote $\bigcup_{Q(j)} X_j$ by $(\bigcup j: Q(j): X_j)$.

Our first task is to make $P0_i$ true initially. The multiple assignment

$$c_i, S_i := 0, W_i$$

suffices for this because $true \Rightarrow wp(\text{“}c_i, S_i := 0, W_i\text{”}, P0_i)$.

Our next task is choose a guard $\beta_i$ for the loop. $\beta_i$ must satisfy $\neg\beta_i \wedge P0_i \Rightarrow R_i$, and an obvious choice for the guard is $c_i \neq diam(i)$. Unfortunately, due to the Local Topology Assumption, this guard is not computable at site $i$ because $diam(i)$ may not be known there. We return to this problem later. For the meantime, we use $c_i \neq diam(i)$.

Finally, we develop the body of the loop. Execution of the loop body, in a state in which $\beta_i$ and $P0_i$ are true must reestablish the loop invariant and make progress towards termination. Progress can be made by increasing $c_i$ (see $P0_i$). In order to reestablish $P0_i$, values must be added to $S_i$. Since $wp(\text{“}c_i, S_i := c_i + 1, S_i \cup Q_i^{c_i+1}\text{”}, P0_i)$ is

$$S_i \cup Q_i^{c_i+1} = (\bigcup j: 0 \leq j \leq c_i + 1: Q_i^j) \wedge 0 \leq c_i + 1,$$

which is implied by $P0_i \wedge c_i \neq diam(i)$, the assignment

$$c_i, S_i := c_i + 1, S_i \cup Q_i^{c_i+1} \tag{3.1.0}$$

suffices.

It remains to compute $Q_i^{c_i+1}$ from values local to site $i$. By definition, $x \in Q_i^{c_i+1}$ if and only if

(i) $x \in Q_k^{c_i}$ for some $k \in pred_i$, and

(ii) there is no $k' \in pred_i$ for which $x \in Q_{k'}^{c'} \wedge c' < c_i$.

That is, $x \in Q_i^{c_i+1}$ if there is a path $(p, \ldots, k, i)$ of length $c_i + 1$ where $x \in W_p$ and there is no path from $p$ to $i$ (through $k$ or any other predecessor of $i$) with length less than $c_i$. Thus[4],

$$\begin{aligned} Q_i^{c_i+1} &= (\bigcup k: k \in pred_i: Q_k^{c_i}) - (\bigcup k': k' \in pred_i: (\bigcup j: 0 \leq j < c_i: Q_{k'}^j)) \\ &= (\bigcup k: k \in pred_i: Q_k^{c_i}) - (\bigcup j: 0 \leq j \leq c_i: Q_i^j) \\ &= (\bigcup k: k \in pred_i: Q_k^{c_i}) - S_i. \end{aligned}$$

(The last step follows from $P0_i$.) Substituting this expression for $Q_i^{c_i+1}$ into (3.1.0), the assignment statement for the loop body becomes

$$c_i, S_i := c_i + 1, S_i \cup ((\bigcup k: k \in pred_i: Q_k^{c_i}) - S_i)$$

which simplifies to

$$c_i, S_i := c_i + 1, S_i \cup (\bigcup k: k \in pred_i: Q_k^{c_i}). \tag{3.1.1}$$

[4] The symbol $-$ denotes set difference.

This assignment can be executed at site $i$ only if sets $Q_k^{c_i}$ for all $k \in pred_i$ are known to $i$. We therefore assume the existence of a routine $Acquire_i$, defined by

$\{true\}$

$Acquire_i$

$\{(\forall k: k \in pred_i: V_i[k] = Q_k^{c_i})\}$.

Then, the program we have developed thus far is obtained by rewriting (3.1.1) using $V_i[k]$ in place of $Q_k^{c_i}$:

$c_i, S_i := 0, W_i;$
$\{P0_i\}$
**do** $c_i \neq diam(i) \rightarrow \{c_i \neq diam(i) \wedge P0_i\}$
$Acquire_i;$
$\{P0_i \wedge (\forall k: k \in pred_i: V_i[k] = Q_k^{c_i})\}$
$c_i, S_i := c_i + 1, S_i \cup (\bigcup k: k \in pred_i: V_i[k])$
$\{P0_i\}$
**od**
$\{P0_i \wedge c_i = diam(i)\}$
$\{R_i\}$

### *3.2. A computable guard*

From the W-Assumption and the definition of $Q_i^t$ we have

$$(t > diam(i)) \Leftrightarrow (Q_i^t = \emptyset) \tag{3.2.0}$$

Computation of $Q_i^{c_i}$ by site $i$ is accomplished by maintaining a set-valued variable $T_i$ defined by

$$P1_i: \quad T_i = (\bigcup j: 0 \leq j < c_i: Q_i^{c_i}).$$

By making $P1_i$ an invariant of the loop, $Q_i^{c_i}$ can be computed during each iteration by evaluating $S_i - T_i$ because

$$P0_i \wedge P1_i \Rightarrow (Q_i^{c_i} = S_i - T_i). \tag{3.2.1}$$

Finally, $P0 \wedge c_i > diam(i) \Rightarrow R_i$. Thus, we are free to choose $c_i \leq diam(i)$ as $\beta_i$ the guard for the loop. This is convenient, because, according to (3.2.0), $\beta_i \Leftrightarrow Q_i^{c_i} \neq \emptyset$. And, from (3.2.1) we get $\beta_i \Leftrightarrow S_i \neq T_i$. This last formulation has the additional virtue that it is entirely in terms of variables maintained at site $i$.

Adding statements to the program to establish and maintain $T_i$ and changing the guard as just suggested yields the following program at site $i$:

$c_i, S_i, T_i := 0, W_i, \emptyset;$

$\{P0_i \wedge P1_i\}$

**do** $S_i \neq T_i \rightarrow \{S_i \neq T_i \wedge P0_i \wedge P1_i\}$

$Acquire_i;$

$\{P0_i \wedge P1_i \wedge (\forall k: k \in pred_i: V_i[k] = Q_k^{c_i})\}$

$c_i, S_i, T_i := c_i + 1, S_i \cup (\bigcup k: k \in pred_i: V_i[k]), S_i$

$\{P0_i \wedge P1_i\}$

**od**

$\{P0_i \wedge c_i > diam(i)\}$

$\{R_i\}$

### 3.3. *Implementing* $Acquire_i$

During iteration $t$ of the loop, the sets $Q_k^t$ for all $k \in pred_i$ are obtained by $Acquire_i$. $P1_k$ is an invariant of the loop at site $k$, so $k$ computes $Q_k^{c_k}$ during each iteration simply by evaluating $S_k - T_k$ and broadcasts the value to its successors.

The loop body at site $k$ is executed $diam(k)+1$ times because $c_k$ is initially 0, it is increased by one each iteration, and the body is no longer executed when $c_k > diam(k)$. Thus, if[5] the loop terminates, $diam(k)+1$ values are sent by $k$ to each site in $succ_k$.

Now consider a site $i$, $i \in succ_k$. Because messages sent along link $(k, i)$ are delivered in the order sent (due to VC1 and VC2), the successive values received on that link are $Q_k^0, Q_k^1, \ldots, Q_k^{diam(k)}$. Therefore, we can implement $Acquire_i$ by

**broadcast** $S_i - T_i;$

**cobegin** $\underset{k \in pred_i}{//}$ **receive** $V_i[k]$ **from** $k$ $\{V_i[k] = Q_k^{c_i}\}$ **coend**

$\{(\forall k: k \in pred_i: V_i[k] = Q_k^{c_i})\}.$

Unfortunately, this introduces the possibility of infinite blocking. The **cobegin** terminates only if every **receive** terminates, and a **receive** terminates only if there is a message available for receipt. $Acquire_i$ is executed once per loop iteration, i.e. $diam(i)+1$ times. Therefore, at least $diam(i)+1$ messages must be sent on link $(k, i)$ for each $k \in pred_i$ to prevent infinite blocking at site $i$.

From the definition of *diam* and the fact that $i$ is a successor of $k$, we obtain

$$diam(k)+1 \geq diam(i). \tag{3.3.0}$$

Consequently, if $k$ makes a **broadcast** after completing $diam(k)+1$ iterations, then the total number of messages sent by $k$ is $diam(k)+2 \geq diam(i)+1$ (by (3.3.0)) and infinite blocking at $i$ is avoided. However, this does mean that the number of

[5] 'If' because absence of deadlock and loop termination have not yet been shown.

messages broadcast by $k$ can be greater than the number of messages received by $i$. In this case, some messages on link $(k, i)$ will not be received. These messages contain information already in $S_i$ and therefore can be safely discarded.

Inserting the code for $Acquire_i$ into the program and adding a **broadcast** after the loop yields

$c_i, S_i, T_i := 0, W_i, \emptyset;$

$\{P0_i \wedge P1_i\}$

**do** $S_i \neq T_i \rightarrow \{S_i \neq T_i \wedge P0_i \wedge P1_i\}$

**broadcast** $S_i - T_i$;

$\{S_i \neq T_i \wedge P0_i \wedge P1_i\}$

**cobegin** $\underset{k \in pred_i}{/\!/}$ $\{S_i \neq T_i \wedge P0_i \wedge P1_i\}$

$r_{ki}$: **receive** $V_i[k]$ **from** $k$

$\{P0_i \wedge P1_i \wedge V_i[k] = Q_k^{c_i}\}$

**coend**;

$\{P0_i \wedge P1_i \wedge (\forall k: k \in pred_i: V_i[k] = Q_k^{c_i})\}$

$c_i, S_i, T_i := c_i + 1, S_i \cup (\bigcup k: k \in pred_i: V_i[k]), S_i$

$\{P0_i \wedge P1_i\}$

**od**;

$\{c_i > diam(i) \wedge P0_i \wedge P1_i\}$

**broadcast** $S_i - T_i$

$\{c_i > diam(i) \wedge P0_i \wedge P1_i\}$

$\{R_i\}$

### *3.4. Termination of loops*

A variant function (called a bound function in [4]) for the loop at site $i$ is

$v$: $diam(i) + 1 - c_i$.

Each iteration of the loop decreases $v$. When $v = 0$, $diam(i) + 1 = c_i$. Thus, by (3.2.0), $Q_i^{c_i} = \emptyset$, and so by (3.2.1) the guard $S_i \neq T_i$ is false and the loop terminates.

### *3.5. Absence of deadlock*

Suppose the process at site $i$ is deadlocked, waiting forever at some receive $r_{ki}$ for a message on link $(k, i)$. By Lemma 1 below, $k$ is also deadlocked. By induction and the finiteness of the system, there exists a cycle of deadlocked processes, each

waiting for a message from the next. For each such link $(k, i)$ we have $c_i = c_k + 1$ because every message sent by $k$ to $i$ has been received and $k$ must therefore be blocked at **receive** $r_{jk}$ following its $c_k + 1st$ broadcast. By transitivity and the fact that the links form a cycle, we conclude $c_i > c_i$, a contradiction. Hence, there is no deadlock.

**Lemma 1.** *If process i is deadlocked at a* **receive** $r_{ki}$, *then k is deadlocked.*

**Proof.** Since no message is forthcoming from $k$, $k$ has terminated or is deadlocked. We assume $k$ has terminated and prove the contradiction $diam(i) > diam(i)$.

$diam(i) \geq c_i$ from $P0_i$ and $S_i \neq T_i$ in $pre(r_{ki})$, and (3.2.0), and (3.2.1).

$c_i = c_k + 1$ because from the postcondition of process $k$, we know that $c_k + 1$ broadcasts were made by $k$ before it terminated, and since $i$ is deadlocked at $r_{ki}$, $i$ must have received all of them.

$c_k + 1 > diam(k) + 1$ from the postcondition of process $k$.

$diam(k) + 1 \geq diam(i)$ due to (3.3.0). □

## 4. Discussion

The strategy we used to derive this distributed algorithm is essentially the 'programming calculus' first proposed for sequential programs in [1]. When non-local values were required (as in computing $Q_i^{c_i+1}$), a **receive** statement was employed and we assumed that the correct values would be received when it executed. A **broadcast** ensured that correct values were available for receipt.

The behavior of the algorithm we derived is not unlike what is observed when a stone is tossed in a pond. A circular wave forms around where the stone entered the pond and expands outward, until it has traversed the entire pond. Whenever the wave passes through an obstruction that penetrates the surface of the pond, another wave is induced—this time, around the obstruction. That wave spreads out until it has traversed the entire pond, causing more waves to be induced as it passes through obstructions, etc. Eventually, after all the waves have traversed the surface of the pond, everything becomes still again. The 'obstructions' in our algorithm are processors; the 'waves' are messages carrying local information stored by every processor encountered. Not surprisingly, we refer to algorithms that work in this fashion as *wave algorithms.*

Wave algorithms have appeared in a number of places in the literature, although to our knowledge the general paradigm has never been discussed. An algorithm attributed to R.G. Gallager in [3] computes network routing information when links are bidirectional, in contrast to our unidirectional links. A network resynchronization procedure based on the wave paradigm, (again for bidirectional links) is described in [2] and [6]. Algorithms to compute partial routing information—in particular, a directed path from one node to another when links are unidirectional—are described

and proved correct in [5] for buffered message-passing and [8] for unbuffered message-passing.

## Acknowledgment

We would like to thank Bowen Alpern, David Gries, Jay Misra, Abha Moitra and Dave Wright for their comments on early drafts of this paper and Carl E. Landwehr for bringing the problem to our attention.

## References

[1] E.W. Dijkstra, *A Discipline of Programming* (Prentice-Hall, Englewood Cliffs, NJ, 1976).
[2] S.G. Finn, Resynch procedures and a fail-safe network protocol, *IEEE Trans. Comm.* **27** (6) (1979) 840–845.
[3] D.U. Friedman, Communication complexity of distributed shortest path algorithms, LIDS-TH-886, M.I.T. (1979).
[4] D. Gries, *The Science of Programming* (Springer, New York, 1981).
[5] A.J. Martin, A distributed path algorithm and its correctness proof, Technical Report AJM21a, Phillips Research Laboratories, Eindhoven, The Netherlands (1980).
[6] A. Segall, Distributed network protocols, *IEEE Trans. Inform. Theory* **29** (1) (1983) 25–35.
[7] A. Tannenbaum, *Computer Networks* (Prentice-Hall, Englewood Cliffs, NJ, 1981).
[8] D. Wright and F.B. Schneider, A distributed path algorithm and its correctness proof, Technical Report TR 83-556, Department of Computer Science, Cornell University, Ithaca, NY (1983).