May 1979 LIDS-P-915

DECENTRALIZED MAXIMUM FLOW ALGORITHMS*

by

Adrian Segall
Faculty of Engineering
Technion Israel Institute of Technology
Haifa, Israel

and

Consultant
Laboratory for Information and Decision Systems
Massachusetts Institute of Technology
Cambridge, Mass. 02139

*Research supported in part by the Defense Advanced Projects Agency (monitored by ONR) under contract ONR/N00014-75-C-1183 and in part by the Office of Naval Research under contract ONR/N00014-77-C-0532.

Submitted for
publication to
"Networks"

DECENTRALIZED MAXIMUM FLOW ALGORITHMS

Adrian Segall

Faculty of Electrical Engineering
Technion - Israel Institute of Technology
Haifa, Israel.

Abstract

The paper presents three algorithms for obtaining maximum flow in a network using distributed computation. Each node in the network has memory and processing capabilities and coordinates the algorithm with its neighbors using control messages. Each of the last two versions requires more sophistication at the nodes than the previous one, but on the other hand employs less communication.

This work was performed on a consulting agreement with the Laboratory for Information and Decision Systems at MIT, Cambridge, Mass., and was supported in part by the Advanced Research Project Agency of the US Department of Defense (monitored by ONR) under Contract No. N00014-75-C-1183, and in part by the Office of Naval Research under Contract No. ONR/N00014-77-C-0532.

1. INTRODUCTION

In this paper we treat the problem of decentralized regulation of the flow of a commodity through a capacitated network. We assume that a controller with a certain computation capability is located at each node of a given network; each controller can measure the amount of flow incoming to and outgoing from the corresponding node, on each of the adjacent links and neighboring controllers are able to exchange information over the link connecting them in the form of control messages. The problem is to design a protocol for each of the controllers, so that the combined algorithm will maximize the total flow through the network , from a given node, called source, to another given node, called sink.

The problem of maximizing network flow using central computation has been widely studied in the literature and many algorithms have been designed for its solution [1] - [5]. On the other hand, technological developments of mini and micro computers have made it possible to introduce relatively sophisticated computation and large memory capabilities in each of the nodes of a transportation or communication network. Decentralized computation provides a serious advantage over the centralized one, mainly because of enhanced survivability and reliability, and because it saves transmission requirements of status information and of command information towards and from the central controller. As such, developing efficient and reliable decentralized algorithms to perform various network functions is of major importance for network design. The research on decentralized network algorithms is just at its starting stages, but several such algorithms have already been developed and validated. Among these we may mention here algorithms for minimum delay routing without and with changing topology [6] - [9], shortest path [10] - [11] and minimum spanning tree [12] - [14].

A pioneering work on a decentralized algorithm for maximum flow can be found in [15], where a decentralized version of the Ford-Fulkerson algorithm is introduced. In this paper we present decentralized versions of three well-known maximum flow algorithms: Ford-Fulkerson [1] (a simpler version than in [15]), Edmonds-Karp [2] and Dinic [3]. In the order presented, each algorithm requires less communication, but more computation and sophistication at the nodes, than the previous one. This trade-off between communication and computation is a subject of much interest to network designers, and in this context it is interesting to observe that, according to L.G. Roberts [16], "the cost of memory and switching has fallen by a factor of 30 compared to transmission costs over the last nine years ...".

2. GENERAL MODEL

Consider a network (N,E) where N is the set of nodes, $E \subset N \times N$ is the set of directed edges and assume that the network has no self-loops and no parallel edges. Two nodes $s, t \in N$ are specified, where s is the source and t the sink. Without loss of generality we may assume that $(j,s) \notin A$ and $(t,j) \notin A$ for all $j \in N$. To each edge $(i,k) \in E$ we attach an integer-valued capacity c_{ik}. If the capacities are not integers but commensurable, an appropriate scale factor will provide an equivalent problem with integer-valued capacities. If $(i,j) \in E$ or $(j,i) \in E$, then node i is said to be adjacent to node j and the corresponding edge is said to be incident to node i.

A feasible flow f is an assignment of an integer f_{ij} to each edge (i,j) such that $0 \leq f_{ij} \leq c_{ij}$ for all $(i,j) \in E$ and such that for all $i \in N - \{s,t\}$ we have

$$\sum_{j:(i,j)\in A} f_{ij} - \sum_{j:(j,i)\in A} f_{ji} = 0$$

Each node i has certain memory, processing and transmission capabilities and is assumed to know and to be able to update the flows f_{ij} and f_{ji} on all edges incident to it. In addition, if $(i,j) \in E$, then nodes i and j are assumed to be able to send to each other control messages that are correctly received at the other node within arbitrary, finite, non-zero time. We may note that this assumption does not preclude that individual control messages are received in error because of noise say, provided that there is an error control and retransmission protocol that will assure error detection and appropriate retransmission until the message is correctly received. Observe also that our assumption implies that if $(i,j) \in E$, then control messages can be sent over it in both directions, even if $(j,i) \notin E$.

The total flow F of a feasible flow f is defined as the net flow from source to sink, namely

$$F = \sum_{j:(s,j)\in E} f_{sj} = \sum_{j:(j,t)\in E} f_{jt} \quad .$$

Our goal is to design a decentralized protocol whereby each node i will make computations and decisions based upon local knowledge of f_{ij} and f_{ji} and upon control messages received from adjacent nodes, to achieve a flow f for which the total flow is maximum.

In the following sections we present three decentralized algorithms for achieving maximum flow. They are decentralized versions of the well-known maximum flow algorithms of Ford-Fulkerson (FF), Edmonds-Karp (EK) and Dinic, respectively. All algorithms are based on flow updating cycles started by the sink-node t, propagating in the network while seeking a flow augmenting path from source to sink and terminating back at the sink. At the time of the termination of a cycle, node t knows that all nodes in the network have completed their part of the previous cycle and are ready to perform a new cycle, so that node t can indeed start a new updating cycle. The protocol also insures that at the completion of the cycle that finally achieves maximum flow, the sink node will be informed that no more cycles are necessary. The three algorithms differ in the procedure for finding augmenting paths. In the first algorithm (based on the FF algorithm), paths are searched and found in a random fashion, employing a decentralized protocol similar to the one used in [8], [9] for adaptive routing. In the second algorithm (based on EK), the augmenting path is always one with smallest number of arcs and to achieve this in a decentralized way, we employ simplified versions of algorithms proposed in [10] and [8]. Finally, in the third algorithm the procedure consists of two kinds of actions as in Dinic [3]: first the nodes find the set of all possible paths with smallest number of arcs, then one augments all

possible paths out of this set, and the procedure is repeated for longer and longer paths.

We shall need now several notations and definitions. For the purpose of simplicity, if $(i,j) \in E$ and $(j,i) \notin E$, then we shall include (j,i) in E and take $c_{ji} \equiv 0$, $f_{ji} \equiv 0$. For a given flow f, let us define for each edge the quantity

$$\rho_{ij} = c_{ij} - f_{ij} + f_{ji} \quad .$$

It can be considered as the available extra-capacity in the direction from i to j of the pair $(i,j),(j,i)$ in the presence of flow f. In particular, additional flow can be pushed from i to j if and only if $\rho_{ij} > 0$. A pair $(i,j),(j,i)$ such that $\rho_{ij} > 0$ or $\rho_{ji} > 0$ (or both) will be called a <u>link</u>, and if $\rho_{ij} > 0$, we shall say that the link is <u>outgoing from</u> i and <u>incoming to</u> j. Observe that these definitions are flow dependent and that if $\rho_{ij} > 0$ and $\rho_{ji} > 0$, then the link is both outgoing from and incoming to i (as well as from and to j). For a given node i, we shall use the notation $[i,j]$ for the pair $(i,j),(j,i)$ and also:

$$I_i = \{k \mid \rho_{ki} > 0\} \ ,$$

$$O_i = \{k \mid \rho_{ik} > 0\} \ .$$

Given any two nodes i,k in the network N, an <u>augmenting path</u> from i to k is defined as a series of distinct nodes, $i = i_o, i_1, i_2, \ldots, i_m = k$ such that for all $n = 0,1,\ldots,(m-1)$ there is a link outgoing from i_n and incoming to i_{n+1}. The quantity $\rho = \min \rho_{i_n, i_{n+1}}$, where the minimum is over $n = 0, \ldots (m-1)$, will be called the <u>available capacity</u> of the augmenting path. If there is at least one augmenting path from i to k, we say that <u>i has access to k</u> and <u>k is accessible from i</u>. We denote by A_r the set of nodes that have access to sink at the time τ_r of starting the r-th cycle, and by L_r the set of links connecting any two nodes in A_r at time τ_r.

In each of the following sections, we shall first describe the general distributed protocols corresponding to each of the versions, then indicate the exact algorithm to be performed by each node as a part of the protocol and finally prove that the protocols indeed achieve maximum flow.

3. VERSION 1

The protocol described in this section is the decentralized version of the Ford-Fulkerson labeling procedure. As said before, the algorithm consists of a series of cycles started by and terminated at the sink node t. We assume that before the first cycle starts, a feasible initial flow exists in the network, e.g. $f_{ik} = 0$ for all edges and also all nodes i are in state S1 with the same cycle number, $n_i = 0$ for all i. A cycle consists of two phases. Phase A is started at sink and propagates into the network. During this phase augmenting paths from network nodes to sink are being built in such a way that all paths belong to a single directed tree rooted at sink. All nodes that have access to the sink t will eventually enter the tree, and therefore the source node s will also join the tree, provided that the existing flow is not maximal. Phase B of a cycle propagates over the tree, from its leaves towards the sink. During this phase, the nodes along the source-sink augmenting path update the flows (or equivalently the quantities ρ_{ik}) and all other nodes make no flow changes. However, they participate in phase B in order to complete their part of the cycle and to finally inform the sink that the present cycle has been terminated. By the time the propagation of phase B arrives at the sink, all nodes in the tree have had completed their part of the current cycle, and therefore the sink knows that a new cycle can be started.

To accomplish its part of the protocol, each node i keeps the following set of variables:

- n_i - cycle identification number (values 0, 1);
- a_i - source-sink augmenting path identification (values, 0, 1);
- d_i - available capacity of augmenting path from i to sink (strictly positive integer-valued, except that $d_t = \infty$);

p_i - pointer to next node on augmenting path from i to sink (called father);

$N_i(k)$ - for each adjacent node k (values 0, 1).

The meaning of $N_i(k)$ will be explained presently. The propagation of the cycle is achieved using control messages sent by nodes to their adjacent nodes and having the format MSG(n,d,a), where, if i is the sending node, then $n = n_i$, $d = d_i$, $a = a_i$. The variables n and n_i serve to distinguish a given cycle from the previous one. The variable a_i takes on normally the value 0; it changes to 1 if node i finds out that it belongs to the current augmenting path from source to sink. If $a_i = 0$, the quantity d_i denotes the available capacity of an augmenting path from i to sink. If $a_i = 1$, then d_i indicates the available capacity of the augmenting path from source to sink and it is exactly the amount of flow pushed from source to sink during the current cycle.

We next proceed to describe the propagation of a flow increasing cycle in the network and the actions taken by the nodes to participate in such a cycle (the numbers in parentheses refer to the Tables). Just before the cycle is started, all nodes in the network are in state S1 with identical cycle numbers $\{n_i\}$. A cycle is started by the sink t when it performs the transition T12 (cf. Fig. 1) from state S1 to state S2 (I.51). At this time it sets the identification number of the new cycle n_t as the binary complement of the number of the previous cycle (I.52) and sends a control message on each of its incoming links (I.53). The new cycle identification number will be carried by all control messages of the present cycle. An arbitrary node i performs its part of phase A when it receives the first message belonging to the present cycle (I.13). At this time it updates its own cycle number n_i (I.17), it chooses the originator of this message ℓ

as father p_i (I.18) and sets the available capacity to sink on the chosen path d_i to the minimum of d_ℓ (received in the message as d) and of the available capacity $\rho_{i\ell}$ on link (i,ℓ). Then, to propagate phase A, it sends a control message over all incoming links, but not to its father p_i (I.20). It is easy to see that, as proven in the Appendix, all nodes having an augmenting path to sink will indeed perform T12 and the links $\{(i,p_i)\}$. will form a tree rooted at sink.

Having completed its part of phase A, node i waits for phase B, and this will be performed when node i had received a control message of this cycle over all of its incoming links. Since node i is not interested in messages coming on links that are not incoming, such messages are immediately "bounced" by node i (I.4, I.5). To recognize the situation that will enable it to perform phase B, node i stores the cycle number of any received message from ℓ in $N_i(\ell)$ (I.3) and will indeed perform phase B when $N_i(k) = n_i$ for all incoming links (I.21). Also, if in the meantime it receives a message with a = 1, node i recognizes that it belongs to the source-sink augmenting path for this cycle, performs the appropriate flow increase (I.6 - I.10) and during transition T21, which is exactly phase B, it augments the flow on the link (i,p_i^*) (I.22 - I.24). In any case it sends during T21 a control message to p_i, thus informing it of the completion of its part of the cycle. It is easy to see (cf. Appendix) that this information will propagate downtree and by the time the sink node t performs T21 (I.54), all nodes on the tree will have completed their part of the present cycle. If an augmenting path has been found and the sink still has incoming links, then a new cycle will be started (I.52); otherwise, maximum flow has been achieved. Finally, observe that the source node s performs the same operations as all other nodes, except that it has no incoming links and that it is always on the source-sink augmenting path (I.27 - I.42).

The decentralized protocol described above indeed achieves maximum flow. To prove this, it is sufficient to show that for any cycle started by the sink node at time τ_1 say:

- the cycle will be completed in finite time;
- if the flow is not maximal at time τ_1, the cycle will increase the flow by a strictly positive integer amount;
- if the flow is maximal at time τ_1, the cycle will not change the flow;
- if at the end of the cycle the flow is not maximal, a new cycle will be started by sink at that time.

The exact proofs of these properties are given in the Appendix.

4. VERSION 2

Edmonds and Karp [2] have shown that if we insist that the chosen augmenting path from source to sink has minimum number of links, then the number of augmentations is $O(|N|^3)$, as opposed to the Ford-Fulkerson algorithm where this number can be bounded only by the value of the maximal flow. We shall now present the decentralized version of the Edmonds-Karp algorithm.

As in Version 1, the present protocol consists of consecutive cycles, started by and completed at the sink node t. A cycle consists of two phases, the first one forming the tree of shortest paths to the sink and the second one propagating downtree to finally inform the sink of the completion of the cycle while increasing the flow on the augmenting source-sink path. Here the finite-state machine for the node needs three states (see Fig. 2): the state S_i of a node i is S0 before it hears of a new cycle, it is S1 whenever the node participates in the first phase of a cycle, namely while it looks for its shortest path to sink, and S_i = S2 after the node has found this path and until it completes the second phase of the cycle. This version differs from Version 1 mainly in the procedure for the node to choose its father p_i, namely during the time the node is in state S1. In Version 1, node i chooses as p_i the link over which it receives the first message of the present cycle. If the delays on all links were identical, then clearly Version 1 would also provide the link corresponding to the shortest path. Since this is not the case however, the idea of the procedure here is to have the protocol act as if the delays were identical. Explicitly, during the phase of looking for the shortest path (i.e. in state S1), a node i sequentially learns (via control messages from its neighbors) whether or not the sink is at distance 1, 2, 3, etc. from it. Suppose, for example, that node i has already learned that the sink is not at distance 1 or 2. If one of the nodes, k say, such that $[i,k]$ is outgoing from i, reports

to i that k is at distance 2 from sink, then node i knows that its shortest path to sink is via k and is 3 links long. The protocol provides this information by requiring each node i to keep the following variables in addition to n_i, d_i, p_i, which have the same purpose as in Version 1:

$d_i = \infty$ - if shortest path has not been found yet; otherwise same meaning as in Version 1;

z_i - distance from node i covered by the algorithm until now (values $0,1,\ldots|N|$);

$M_i(k)$ - for each adjacent node k, counting the number of messages received from k during the present cycle; it is exactly the last z_k reported by k to i and received at node i, (values $0,1,\ldots,|N|$);

$D_i(k)$ - for each adjacent node k (values $0,\infty$).

The meaning of $D_i(k)$ will be explained presently. The quantity $M_i(k)$ is initialized to (-1) at the end of the previous cycle (II.36) and is incremented (II.4) every time node i receives a message from k. The counter z_i is set to 0 when node i enters the present cycle (II.22), incremented to 1 after having received at least one message from all outgoing links, i.e. after $M_i(k) \geq 0$ for all such links (II.24), (II.27), incremented to 2 after $M_i(k) \geq 1$ and so on (II.25), (II.29). Every time z_i is incremented, a control message is sent over all incoming links, to inform the neighbors that i has covered all nodes that are z_i or less links away. Node i declares that it has found the link (i,p_i) providing the shortest path to sink whenever it has received from p_i a message with $d \neq \infty$ (II.5), (II.6), (II.27), and it is ready to increment z_i to $z_{p_i} + 1 = M(p_i) + 1$ (II.27). At this time it performs the transition to state S2 and from here on acts as in Version 1.

In order to validate Version 2, it is sufficient to show that for all nodes i that have access to the sink at the beginning of a cycle, the link (i,p_i) as chosen by the algorithm during this cycle, provided the shortest augmenting path to the sink. This is because, except for the phase of choosing (i,p_i), the algorithm acts as in Version 1 and that version would choose (i,p_i) as providing the shortest path if control message delays were identical on all links. Therefore, since validation of Version 1 was proved for arbitrary control message delays, the case under consideration here becomes a special case.

Lemma 4.1

For each node i with acces to sink, the link $[i,p_i]$ at the time when i performs transition T12, corresponds to the augmenting path with smallest number of links connecting i to sink.

Proof

We proceed by induction. With the notations at the end of Section 2, consider all nodes in A_r that have an augmenting path with m links or less to sink. Suppose that these nodes satisfy the statement of the Lemma. Then a node i whose distance to sink is $(m+1)$ must have an outgoing link to a node k with distance m and no outgoing link to nodes with distance less than m. The link $[i,k]$ provides the shortest augmenting path from i to sink, and when k performs T12 it will send its $(m+1)$-st message to node i with $d \neq \infty$. This will make $d_i \neq \infty$, and when node i will also have received at least $(m+1)$ messages from its other outgoing links, it will perform T12 with $p_i = k$.

5. VERSION 3

Dinic's algorithm [3] achieves maximum flow in $O(|N|^2)$ augmentations by first finding the set of all shortest augmenting paths from source to sink in the subnetwork (A_r, L_r), and then looking for augmenting paths only in this subnetwork. For a given flow, the union of all shortest source-sink paths is called the referent of the network and after exhausting all paths in one referent, a new referent with longer paths will be searched for. Suppose that for the given flow under consideration, the shortest source-sink path has m links. Then the referent contains $(m+1)$ levels, where t is in level 0, the source s is in level m, and an arbitrary node $i \in A_r$ is in level j if its shortest path to sink has j links, and if there is an incoming link $[\ell,i]$ to i such that ℓ is in level $(j+1)$ of the referent. In this case link $[\ell,i]$ belongs to the referent (for details see [3]).

As in the previous sections, the distributed version of Dinic's algorithm is composed of a series of cycles. Each cycle here consists of first finding the referent (Part A in Table 3), and then of a series of subcycles during which augmenting paths in this referent are searched for. During the first part, each node finds out if itself and which of its adjacent links belong to the referent. A node i in the referent compiles the sets R_{in} and R_{out} of nodes that are one level above it and one level below it, respectively. Only those nodes (and the appropriate links) will participate in the second part of the cycle, during which source-sink augmenting paths belonging to the referent will be obtained in the same way as in Section 3 of this paper. (Part B in Table 3). We therefore have to describe here only the first part of a cycle, i.e. determining the referent, which is done as follows. First a node will find its distance from sink in the same way as in

Section 4, so that in state S0 and S1 it will behave as in Table 2. The difference is in state S2, because we require that a node will positively know whether it belongs to the referent or not at the time of performing T20, and if it does, which of its incoming and outgoing links are in R_{in} and R_{out}, respectively. If a node i is at distance j from the sink, then it will belong to level j of the referent if indeed it enters the referent. In this case, the adjacent links that will enter the referent will be outgoing links to the nodes distanced $(j-1)$ hops from sink and incoming links from nodes distanced $(j+1)$ hops from sink. Observe that in Section 4, if $D_i(k) = 0$, namely if a message with $d \neq \infty$ has been previously received from k, then this means that k has found its distance from sink, which is exactly equal to $M_i(k)$. Now, node i finds out its distance from sink in transition T12. Therefore any $k \in O_i$ such that $D_i(k) = 0$, $M_i(k) = z_i$ (here we mean the value of z_i just before the transition) will positively enter the referent if i does and all links $[i,k]$ will also enter the referent in the direction from i to k. We indicate this in (III.3). In addition, if such a node k is also in I_i, node i will not send a message to it while performing T12 (as in Version 2), but rather when performing T20 (III.5), (III.6). By that time, i will know whether it has entered the referent. Node i will actually enter the referent by command received over an incoming link (III.2) and performing T20 without receiving such a command is the information that node i did not enter the referent. The variables a_i and a play here the same role as in Sections 3,4, except that here they indicate that node i belongs to the referent, whereas previously they indicated that i is on the chosen source-sink path.

From the description of the algorithm it is clear that Part A of a cycle of the distributed algorithm here will give the same result as the construction of a referent in [3], and this validates our algorithm. Details of the proof are essentially identical to the proofs in the Appendix and therefore deleted.

6. Summary

We have described distributed versions of three well-known maximal flow algorithms. Probably other existing algorithms [4], [5] can be decentralized using similar ideas. As mentioned before, the processing requirements are simpler in Version 1, become more complicated in Version 2 and even more so in Version 3. On the other hand, as presently seen, the communication requirements are reduced.

For integer valued capacities, Version 1 needs a maximum of F_{max} augmentations, where F_{max} is the value of the maximal flow. Since the algorithm requires a maximum of 1 message/edge (in each direction) for each augmentation, the communication requirement is $O(F_{max})$ messages/edge. Version 2 requires $O(|N|^3)$ augmentations, and since the maximum distance to sink is $|N|$, each augmentation requires no more than $|N|+1$ messages/edge. Hence the communication requirement is no more than $O(|N|^4)$ messages edge. Finally in Dinic's algorithm [3] the number of required cycles is no more than $|N|$. In order to find the referent one needs no more than $|N|+1$ messages/edge as in Version 2, and the number of possible subcycles is bounded by $|E|$, since each subcycle saturates at least one link. For each subcycle one needs no more than 1 message/edge as in Version 1, and hence the communication requirements are bounded by $O(|N| \cdot (|E| + |N|)) = O(|N||E|) \leq O(|N|^3)$ messages/edge.

Acknowledgement

The author would like to thank Prof. S. Even and Dr. O. Kariv for many stimulating discussions regarding the algorithms of this paper.

APPENDIX: Validation of Version 1

In this Appendix we shall prove that the protocol proposed in Section 3 indeed achieves maximum flow. The other Versions have been validated in the body of the paper. We shall first need several definitions and notations. Recall that a cycle is started by the sink t when it performs T12 and is completed when sink performs T21. Let τ_r be the time when the r-th cycle is started and τ_{r+1} the time of its completion, which is the same as the time of starting the (r+1)-st cycle (see I.56). We say that the network (N,E) is 0-relaxed at time τ_r if all nodes $i \in N$ are in state S1, if no messages are in transit and if all nodes i in A_r have cycle number $n_i = 0$. Similarly for 1-relaxed. Also, in order to specify the value of a variable at a given time τ, we put τ in parentheses, e.g. $n_i(\tau)$ denotes the value of n_i at time τ.

Suppose that the network is 0-relaxed at time τ_r and a cycle is started by sink at this time, i.e. sink performs T12 and changes n_t from 0 to 1. In order to see exactly what happens within a cycle, let us assume that whenever (I.54) will hold (if at all), instead of starting a new cycle after performing T21 as required by (I.56), the sink will stop. We shall show that indeed (I.54) will hold in finite time after τ_r, at time τ_{r+1} say, and at that time the network will be 1-relaxed. We shall also see the operations performed by the nodes during the period from τ_r to τ_{r+1} and that, unless the flow was maximal at time τ_r, the cycle will have strictly increased the flow by an integer amount. Then, by induction on the cycles, maximum flow will indeed be reached within a finite number of cycles.

Lemma 1

Suppose that at time τ_r the network is 0-relaxed and sink performs T12 (while changing n_t from 0 to 1). Suppose that if after time τ_k the sink

performs T21, it will not start a new cycle, but will rather stop. Then, with the notations of Section 2,

(a) each node i must change n_i from 0 to 1 before it generates any message;

(b) each node $i \in A_r$ will perform T12 at most once; nodes not in A_r perform no transition;

(c) all generated messages carry $n = 1$;

(d) each node $i \in A_r$ will perform T21 at most once;

(e) no more than one message is sent on each link of L_r in each direction. No messages are sent on links not in L_r.

Proof

Observe that by the assumption of the Lemma, at time τ_r all nodes $i \in A_r$ have $n_i = 0$, so that (a) is clear from (I.13), (I.17). Now we prove (b) - (c) by a common induction. For any instant $\tau \geq \tau_r$, suppose that (b) - (c) hold until time $\tau-$. We shall show that whatever happens at time τ will not violate (b) - (c). Suppose the first part of (b) is violated at time τ, i.e. node i performs T12 for the second time. Then (I.13) says that it receives at this time a message with $n = 0$, which must have been generated before time $\tau-$, violating (b) before $\tau-$. Since messages are sent by i either to $k \in I_i$ (I.20), or on links from which messages have been received ((I.5) and (I.25)), nodes not in A_r receive no messages and hence perform no transitions, completing the proof of (b). Now suppose node i generates a message with $n = 0$ at time τ, violating (c). Then it must have $n_i(\tau) = 0$ violating (b) before or at time τ.

Now (d) is clear, since T12 and T21 must alternate (Fig. 1). In order to prove the first part of (e), we suppose again that it holds through-

out the network until time $\tau-$ and show that at time τ it cannot be violated. First observe that if condition (I.4) holds at time $\tau-$, then by (c) we have $n(\tau-) = n_i(\tau-) = 1$, so that i must have performed T12 earlier. Therefore if (I.4) holds, then $\ell \neq p_i$, since otherwise p_i must have sent two messages to i violating (e) before $\tau-$. Now node i sends a message to p_i in T21, a message to each $k \in I_i$, $k \neq p_i$ in T12 and a message to each $k \notin I_i$, $k \neq p_i$ in (I.5), so that the first part of (e) holds. The second part of (e) follows from the second part of (b).

□

Lemma 2

Suppose the same assumptions as in Lemma 1 hold. Then

(a) each of the nodes in A_r and only those nodes will perform T12 exactly once;

(b) the set A_r and the links (i,p_i) form a tree spanning (A_r,L_r) and rooted at the sink node t;

(c) each node $i \in A_r$ will perform T21 exactly once;

(d) the last node to perform T21 is the sink and at that time, τ_{r+1} say, the network is 1-relaxed and $A_{r+1} \subseteq A_r$.

Proof

(a) At time τ_r all nodes i in A_r are in state S1 with $n_i = 0$. At time τ_r sink performs T12, changes n_t from 0 to 1 and sends a message with $n = 1$ over incoming links. Therefore the nodes i at the other end of these links will eventually receive a message with $n = 1$, and unless they have performed T12 before, they will perform T12 now (I.13) while changing n_i from 0 to 1. Each such node j will send a message with $n = 1$ on all of its incoming links, except to p_j which has already performed T12, so that by induction, all nodes that have at time τ_r at least one augmenting

path to sink will perform T12 and change n_i from 0 to 1. The above reasoning also shows that only nodes in A_r will perform T12.

(b) The graph formed by the links (i,p_i) clearly spans (A_r,L_r). If ℓ is chosen by i as p_i as in (I.18), then ℓ must have performed T12 before i, and hence this graph is loop-free. Since each i has a unique p_i, the graph is a tree. Since there must be at least one node i such that $p_i = t$, and t has no "father" p_t, the tree is rooted at t.

(c) Consider first a node i which is a leaf of the tree (i.e. $\nexists k$ s.t. $p_k = i$). Let us look at an arbitrary node k such that $k \in I_i$, $k \neq p_i$, to which i sends a message when performing T12 (I.20). When this message arrives at k, it must hold that $n = n_i$, otherwise k will choose i as p_k violating the fact that i is a leaf. Therefore, if $k \notin I_i$, then (I.4) will hold and k will send a message to i. On the other hand, if $k \in I_i$, then k has sent a message to i while performing T12. Therefore i will eventually receive messages from all $k \in I_i$, (I.21) will hold, i will perform T21 and send a message to p_i. Similarly, by induction, we can show that each node in A_r will perform T21, this action propagating down-tree.

(d) The reasoning in (c) shows that the sink t is the last to perform T21. If a node $i \notin A_r$, then it and all nodes to which it has access (which are also not in A_r) performed no transitions, hence remained in state S1 and had no change of flow, implying that they are not in A_{r+1} either. Finally, if $i \in A_r$, then it has received all messages on links $[i,k]$ such that $k \in I_i$ before performing T12. If $k \notin I_i$, then $i \in I_k$ and k has received a message from i before k performed T21. But i can send such a message only when receiving a message from k. Therefore i receives all messages sent to it before sink performs T21, so that there are no messages in transit at time τ_{r+1} and hence the network is 1-relaxed.

□

Theorem

Let F be the total flow entering the sink. In the protocol indicated in Section 3 and Table 1, we have for all cycles

$$F(\tau_{r+1}) \geq F(\tau_r) \qquad (A.1)$$

with equality if and only if $F(\tau_r)$ is maximal. If the latter holds, then the sink will stop after the $(r+1)$-st cycle or possibly even before this cycle. Therefore maximal flow will be reached in a finite number of cycles.

Proof

Clearly the flow is maximal at τ_r if and only if $s \notin A_r$. If $s \in A_r$, it will eventually enter the tree, perform T12 and immediately T21 while sending $MSG(n_s, d_s, a_s = 1)$ to p_s (I.41). When $k = p_s$ receives it, it sets $d_k = d_s$ and $a_k = 1$ and when performing T21 sends $MSG(n_k, d_k, a_k = 1)$ to p_k. By induction, the nodes of the entire source-sink path on the tree will perform similar operations, while also changing the flow (or equivalently the available extra-capacities $\rho_{i\ell}$). Therefore (A.1) holds with inequality.

On the other hand if $s \notin A_r$, either I_t at time τ_r is empty in which case the sink starts no new cycle or else the cycle is started, but s does not enter the tree and the cycle will be completed with $a_t = 0$. In either case the flow is maximal and the sink stops triggering cycles (I.57).

□

TABLE 1: Algorithm for Version 1

1a) Algorithm for Node $i \in N - \{s,t\}$

Note: Message MSG(n,d,a) received from node ℓ is delivered to the algorithm in the form MSG(n,d,a,ℓ).

I.1 Message Handling

I.2 When receiving MSG(n,d,a,ℓ), execute:

I.3 $N_i(\ell) \leftarrow n$;

I.4 if $n = n_i$, $\ell \notin I_i$ (comment: $\ell \neq p_i$, $a = 0$),

I.5 then send $MSG(n_i,d_i,0)$ to ℓ;

I.6 if $a = 1$ (comment: i is in state S2, $\ell \neq p_i$)

I.7 then $\rho_{i\ell} \leftarrow \rho_{i\ell} + d$,

I.8 $\rho_{\ell i} \leftarrow \rho_{\ell i} - d$,

I.9 $d_i \leftarrow d$,

I.10 $a_i \leftarrow 1$;

I.11 execute Finite-State-Machine.

I.12 Finite-State-Machine Transitions (see Fig. 1)

Note: The Finite-State-Machine is executed until no more transitions are possible.

I.13 T12 Condition 12 $MSG(n \neq n_i,d,a,\ell)$.

I.14 Comment 12 $a = 0$, $a_i = 0$, $\rho_{i\ell} > 0$.

I.15 Action 12 $I_i \triangleq \{k | \rho_{ki} > 0\}$;

I.16 $O_i \triangleq \{k | \rho_{ik} > 0\}$;

I.17 $n_i \leftarrow n$;

I.18 $p_i \leftarrow \ell$;

I.19 $d_i \leftarrow \min(d,\rho_{i\ell})$

I.20 send $MSG(n_i,d_i,a_i)$ to all k such that $k \in I_i$, $k \neq p_i$.

I.21 T21 Condition 21 $\forall k \in I_i$, then $N_i(k) = n_i$.

I.22 Action 21 If $a_i = 1$, then

I.23 $\rho_{p_i i} \leftarrow \rho_{p_i i} + d_i$,

I.24 $\rho_{i p_i} \leftarrow \rho_{i p_i} - d_i$;

I.25 send $MSG(n_i, d_i, a_i)$ to p_i;

I.26 $a_i \leftarrow 0$.

1b) Algorithm for Source Node s

Note: $I_s = \phi$

I.27 Message Handling

I.28 When receiving $MSG(n,d,a,\ell)$, execute:

I.29 if $n = n_s$,

I.30 then send $MSG(n_s, d_s, 0)$ to ℓ;

I.31 execute Finite-State-Machine for s.

I.32 Finite-State-Machine Transitions for s (see Fig. 1)

I.33 T12 Condition 12 $MSG(n \neq n_i, d, a, \ell)$.

I.34 Action 12 $d_s \leftarrow \min(d, \rho_{s\ell})$;

I.35 $n_s \leftarrow n$;

I.36 $p_s \leftarrow \ell$;

I.37 $a_s \leftarrow 1$;

I.38 execute transition T21.

I.39 T21 Condition 21 none.

I.40 Action 21 $\rho_{s p_s} \leftarrow \rho_{s p_s} - d_s$;

I.41 send $MSG(n_s, d_s, a_s)$ to p_s;

I.42 $a_s \leftarrow 0$.

Ic) Algorithm for Sink Node t

Note: $O_t = \phi$; $d_t \equiv \infty$.

I.43 Message Handling

I.44 When receiving MSG(n,d,a,ℓ), execute:

I.45 $N_i(\ell) \leftarrow n$;

I.46 if $a = 1$

I.47 then $\rho_{\ell t} \leftarrow \rho_{\ell t} - d$,

I.48 $a_t \leftarrow 1$;

I.49 execute Finite-State-Machine.

I.50 Finite-State-Machine Transitions (see Fig. 1)

I.51 T12 Condition 12 none.

I.52 Action 12 $n_t \leftarrow \bar{n}_t$ (note: $\bar{n}_t$ is the binary complement of n_t);

I.53 send MSG($n_t, d_t = \infty, a_t$) to all $k \in I_t$.

I.54 T21 Condition 21 $\forall k \in I_t$, then $N_t(k) = n_t$.

I.55 Action 21 $I_t \triangleq \{k | \rho_{kt} > 0\}$;

I.56 If $a_t = 1$ and $I_t \neq \phi$, then $a_t \leftarrow 0$ and perform T12;

I.57 else STOP, maximum flow achieved.

TABLE 2: Algorithm for Version 2

Algorithm for Node $i \in N - \{s,t\}$

II.1 Message Handling

II.2 When receiving MSG(n,d,a,ℓ), execute:

II.3 if $d \neq \infty$, then $D_i(\ell) \leftarrow 0$,

II.4 $M_i(\ell) \leftarrow M_i(\ell) + 1$,

II.5 if $d \neq \infty$, $d_i = \infty$, then $p_i \leftarrow \ell$, $d_i \leftarrow \min\{d, \rho_{i\ell}\}$,

II.6 if $d \neq \infty$, $d_i \neq \infty$, $M_i(\ell) < M(p_i)$, then $p_i \leftarrow \ell$, $d_i \leftarrow \min\{d, \rho_{i\ell}\}$,

II.7 if $n = n_i$, $\ell \notin I_i$, $d \neq \infty$, i is in state S2 or S0

(comment: $\ell \neq p_i$, $a = 0$);

II.8 then send MSG(n_i,d_i,0) to ℓ;

II.9 if $a = 1$ (comment: i is in state S2, $\ell \neq p_i$)

II.10 then $\rho_{i\ell} \leftarrow \rho_{i\ell} + d$,

II.11 $\rho_{\ell i} \leftarrow \rho_{\ell i} - d$,

II.12 $d_i \leftarrow d$,

II.13 $a_i \leftarrow 1$;

II.14 execute Finite-State-Machine

II.15 Finite-State-Machine Transitions (see Fig. 2)

II.16 T01 Condition C1 MSG($n \neq n_i$,d,a,ℓ).

II.17 Action C1 $I_i \triangleq \{k | \rho_{ki} > 0\}$;

II.18 $O_i \triangleq \{k | \rho_{ik} > 0\}$;

II.19 $a_i \leftarrow 0$;

II.20 $n_i \leftarrow n$;

II.21 $d_i \leftarrow \infty$;

II.22 $z_i \leftarrow 0$;

II.23 send MSG(n_i,d_i,a_i) to all $k \in I_i$.

II.24 T11 Condition 11 $\{\forall k \in O_i$, then $M_i(k) \geq z_i\}$ and

$\{[d_i = \infty]$ or $[(d_i \neq \infty)$ and $(z_i < M_i(p_i)]]\}$.

II.25 Action 11 $z_i \leftarrow z_i + 1$;

II.26 send $MSG(n_i, d = \infty, a_i)$ to all $k \in I_i$.

II.27 T12 Condition 12 $\{\forall k \in O_i$, then $M_i(k) \geq z_i\}$ and

$\{d_i \neq \infty\}$ and $\{z_i = M_i(p_i)\}$.

II.28 Comment 12 $a_i = 0$.

II.29 Action 12 $z_i \leftarrow z_i + 1$;

II.30 send $MSG(n_i, d_i, a_i)$ to all $k \in I_i$, $k \neq p_i$.

II.31 T20 Condition 20 $\forall k \in I_i$, then $D_i(k) = 0$.

II.32 Action 20 if $a_i = 1$, then

II.33 $\rho_{p_i i} \leftarrow \rho_{p_i i} + d_i$,

II.34 $\rho_{i p_i} \leftarrow \rho_{i p_i} - d_i$,

II.35 send $MSG(n_i, d_i, a_i)$ to p_i;

II.36 $\forall k \in I_i \cup O_i$, then $M_i(k) \leftarrow -1$, $D_i(k) \leftarrow \infty$;

Note: To save space, we do not indicate explicitly the algorithms for source and sink. The source s acts as all other nodes, except that it has no incoming links, so that it performs T20 immediately after T12 and also $a_s \equiv 1$. The sink t has only transitions T02 and T20 which are identical to T12 and T21 of Version 1. It will stop the algorithm when it will have $a_t = 0$ at the time of performing T20. Then maximum flow is achieved.

TABLE 3: Algorithm for Version 3

Part A: Finding the Referent.

All instructions are exactly as in Table 2, except for the following changes:

III.1 In (II.7) change $\ell \notin I_i$ to $\ell \notin I_i \cup R_{out}$.

III.2 Change (II.9) - (II.13) to: if $a = 1$ (comment: i is in state S2) then $a_i \leftarrow 1$ and include ℓ in R_{in}.

Action 12 should read:

III.3 $R_{out} \triangleq \{k | k \in O_i, D_i(k) = 0, M_i(k) = z_i\}$;

III.4 $z_i \leftarrow z_i + 1$;

III.5 send $MSG(n_i, d_i, a_i, r_i)$ to all $k \in I_i$, $k \notin R_{out}$.

III.6 In Action 20, delete (II.32) - (II.34) and change (II.35) to:

send $MSG(n_i, d_i, a_i)$ to all $k \in R_{out}$.

III.7 In Action 20, add: $n_i' \leftarrow 0$.

Part B: Finding Augmenting Paths

The algorithm is almost identical to Table 1. All variables, states and transitions here will have the same names as in Table 1, except that they will be primed (e.g. n_i', s1', T1'2', etc.) to distinguish them from those of Part A. All instructions are exactly as in Table 1, except for the following changes:

III.8 In (I.4) change $\ell \notin I_i$ to $\ell' \in R_{out}$.

III.9 Change (I.8) to: $\rho_{\ell i} \leftarrow \rho_{\ell i} - d$ and then if $\rho_{\ell i} = 0$, exclude ℓ from R_{in}.

III.10 Delete (I.15), (I.16) from Action 12.

III.11 Change (I.20) to: send $MSG'(n_i', d_i', a_i')$ to all $k \in R_{in}$.

III.12 Change (I.21) to: $\nexists k \in R_{in}$, then $N_i'(k) = n_i'$.

III.13 Change (I.24) to: $\rho_{ip_i} \leftarrow \rho_{ip_i} - d_i$ and then if $\rho_{ip_i} = 0$, exclude ℓ from R_{out}.

III.14 Change (I.57) to: else perform T01.

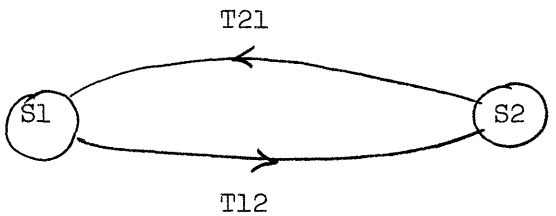

Fig. 1 - Finite-State-Machine for Version 1.

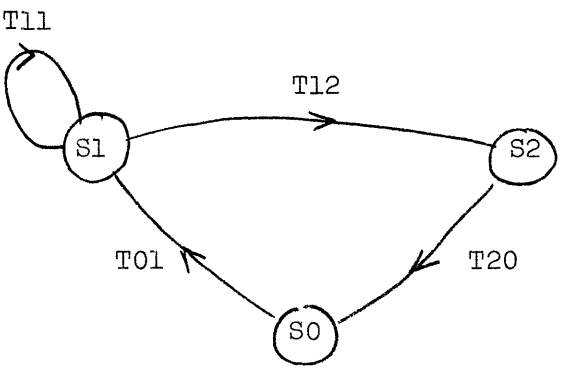

Fig. 2 - Finite-State-Machine for Version 2.

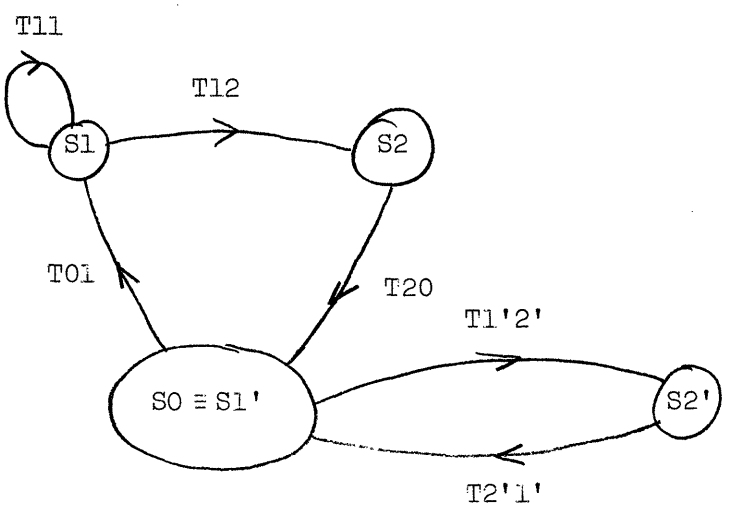

Fig. 3 - Finite-State-Machine for Version 3.

References

[1] L.R. Ford and D.R. Fulkerson, Flows in Networks, Princeton Univ. Press, Princeton, N.J., 1962.

[2] J. Edmonds and R.M. Karp, Theoretical Improvements in Algorithm Efficiency for Network Flow Problems, JACM, Vol. 19, No. 2, pp. 248-264, 1972.

[3] E.A. Dinic, Algorithm for Solution of a Problem of Maximum Flow in a Network with Power Estimation, Soviet Math. Dokl., Vol. 11, pp. 1277-1280, 1970.

[4] A.V. Karzanov, Determining the Maximal Flow in a Network by the Method of Preflows, Soviet Math. Dokl., Vol. 15, pp. 434-437, 1974.

[5] V.M. Malhotra, M. Pramodh Kumar and S.N. Mahashwari, An $O(|\hat{N}|^3)$ Algorithm for Finding Maximum Flows in Networks, preprint.

[6] R.G. Gallager, A. Minimum Delay Routing Algorithm using Distributed Computation, IEEE Trans. Comm., Vol. COM-25, pp. 73-85, January 1977.

[7] A. Segall, Optimal Distributed Routing for Virtual Line-Switched Data Networks, IEEE Trans. Comm., Vol. COM-27, pp. 201-209, Jan. 1979.

[8] A. Segall, P.M. Merlin, R.G. GAllager, A Recoverable Protocol for Loop-Free Distributed Routing, ICC 78, Toronto, June 1978.

[9] P.M. Merlin and A. Segall, A Failsafe Distributed Routing Protocol, submitted to IEEE Trans. Comm.

[10] R.G. Gallager, A shortest Path Routing Algorithm with Automatic Resync, MIT Memo, March 1976.

[11] P.A. Humblet, A Distributed Shortest-Path Algorithm, International Telemetering Conf., Los Angeles, Nov. 1978.

[12] Y. Dalal, Broadcast Protocols in Packet Switched Computer Networks, Ph.D. thesis, Stanford Univ., April 1977.

[13] P.M. Spira, Communication Complexity of Distributed Minimum Spanning Tree Algorithms, Second Berkeley Workshop on Distributed Data Management and Computer Networks, May 1977.

[14] R.G. Gallager, Minimum Weight Spanning Trees, MIT Memo, 1978.

[15] R. Lau, R.M. Persiano and P. Varaiya, Decentralized Information and Control: A Network Flow Example, IEEE Trans. Autom. Control, Vol. AC-17, pp. 466-474, Aug. 1972.

[16] L.G. Roberts, The Evolution of Packet Switching, Proc. IEEE, Vol. 66, pp. 1307-1314, Nov. 1978.