DISTRIBUTED SHORTEST-PATH PROTOCOLS for TIME-DEPENDENT NETWORKS

Ariel Orda
Raphael Rom†

Department of Electrical Engineering
and the S. Neaman Institute
Technion - Israel Institute of Technology
Haifa, Israel 32000

October 1992
Revised May 1994, October 1995

ABSTRACT

This paper addresses algorithms for networks whose behavior changes with time according to known functions. Because the computation depends on the the same functions it attempts to compute, its execution must obey strict timing constraints. When distributed versions of such algorithms are considered, a key difficulty is how to transfer local timing functions among the participating nodes. To that end it is necessary to characterize the parameterization of the functions and accommodate this parameterization in the computation.

In particular, we consider the shortest-path problem in networks in which the delay of the edges changes with time according to continuous functions. We present distributed protocols for finding the shortest and minimum delay path under various waiting constraints. We investigate and analyze protocols that exchange local time-delay functions only within limited intervals yet allow every node to calculate its representation in the shortest path in time for it to be used.

†Also with Sun Microsystems Inc., Mountain View, CA, 94043

I. INTRODUCTION

Network algorithms have been the subject of extensive research for quite some time. These algorithms (which can be centralized or distributed) deal with dynamic systems in which the result of the algorithm depends on the current state of the system.

A class of algorithms that was hardly investigated is that which computes network-functions that are time dependent. Nonetheless, this class of algorithms naturally arises in several networks. A simple example is that of a computer communications network composed of dial-up links each with its individual dialing schedule. Since delays depend on these predetermined schedules, finding the best route for a message from source to destination involves the computation of time-dependent functions. In fact, many networks exhibit a dynamic behavior that, to a very large extent, can be predicted or observed or recorded. Transportation networks (e.g., the railway network) are a straightforward example. Certain types of computer and communication networks belong to this class as well. Aside from the dial-up network mentioned above, consider a mobile network consisting of stations that move in predetermined routes (such as non-geostationary satellites); since the individual orbits are known, the distance between every two stations is a computable function of time. Another example is the well known "rush-hour" behavior in which the dynamic behavior of the network as a function of time of day can be fairly well predicted. Several such problems (mostly in a centralized version) have recently appeared in the literature[1,2,3,4,5,6,7,8].

Time dependent algorithms that run in real time have an additional time constraint imposed on their execution. While general algorithms need not predict their termination time, algorithms for time-dependent networks must always complete the computation for any moment of time before that time arrives. As we shall see, this is a central issue for the distributed computation of these functions.

There is a natural relation between the distributed implementation of control algorithms and the dynamic behavior of networks, since a distributed solution enables constant tracking of changes and corresponding updates. Nonetheless, distributed computation of time varying functions was heretofore hardly addressed.

In this paper we consider time-dependent networks, addressing one of the most basic problems--the shortest path problem--and focus on a distributed solution. Shortest paths algorithms have been the subject of extensive research for many years resulting in a large number of algorithms for various conditions and constraints[9]. Time dependency has been recently investigated (e.g., [6] and the references therein), but not in the context of distributed algorithms.

We present distributed algorithms for finding minimum delay paths in time dependent networks. Our interpretation of this time dependency is that of message traversal delay. We limit ourselves to continuous functions. The execution of the algorithms involves performing calculations on functions of time and exchanging such functions among nodes. The algorithms find minimal-delay paths from all nodes to a given destination for an entire interval of time and enable each node at any time to route messages according to the minimum delay path appropriate for that moment of time. To facilitate the exchange, one of the algorithms presented exchanges functions within a limited time domain, thereby allowing a finite representation of functions or the use of approximations. This limited-time mode of operation is strictly designed in such a way that computation regarding a certain time interval is guaranteed to terminate before this information must be used. This design also considers the time it takes to exchange the function description.

This paper is structured as follows. We first present (section II) a general model and briefly summarize prior results, those that make up the basis of our algorithms. We address in this section a general model for time dependency, a means to incorporate the representation of the delay functions in the calculation, and the model of the distributed computation. In Section III we present the distributed shortest-path algorithms and discuss their properties and characteristics. We present and investigate algorithms that operate on an infinite time horizon and then present and investigate the more practical class of algorithms, namely those that operate within finite time intervals. At the end of this section we remark on how our algorithms can be used when the delay functions only approximate the actual behavior. Concluding remarks are presented in Section IV.

II. MODEL AND PREVIOUS RESULTS

A. Definitions

We consider a directed network $G(V,E,D)$, with $V=\{1,2,...,n\}$ being the set of nodes, $E\subseteq V\times V$ the set of links, and $D=\{d_{ik}(t)\,|\,(i,k)\in E\}$ a set of time-dependent link delays i.e., $d_{ik}(t)$ is a strictly positive function of time defined for $[0,\infty)$ describing the delay of a message over link (i,k) at time t.

Several interpretations of the link delay functions are possible resulting in somewhat different models. We adopt the *frozen link* model[6] in which the delay of a message is fixed at the time a message starts traversing it. Note that in this model non-FIFO behavior is possible. Thus, it may sometimes be preferable to wait a certain amount of time at the sending node before embarking on the link traversal. Such waiting consumes buffer space and may not be permitted by all nodes. We consider the following two network behavior styles:

- *Unrestricted waiting* (UW) in which unlimited waiting is allowed everywhere along the message path through the network.
- *Source waiting* (SW) in which waiting is disallowed everywhere along the message path through the network except at the source node which permits unlimited waiting.

(In [6] it is shown that when waiting is dissallowed altogether, pathological behavior may result, such as shortest paths with loops or even infinite shortest paths.)

To get familiar with these notions consider Figure 1. In Figure 1.a we observe a message departing node i at time t_1 when the link delay is $d_{ik}(t_1)$, meaning that the message arrives at node k at $t_1'=t_1+d_{ik}(t_1)$. Figure 1.b depicts arrival times at node k for several departing times from node i. Notice that a delayed departure at t_3 results in an earlier arrival.

The reader will easily convince himself that to minimize the delay a message arriving at t_1 should depart at t_4 (see Figure 1.c). The point t_4 can be characterized as the rightmost intersection point in $[t_1,t_1+d_{ik}(t_1)]$ between $d_{ik}(t)$ and the 45° chord closest to the origin.

We now define the terminology used in analyzing the above models. Consider a message that arrives at node i at time t, waits for a period of τ and then departs on link (i,k) at time $t+\tau$. This message arrives at node k at time $t+[\tau+d_{ik}(t+\tau)]$. We define $D_{ik}(t,\tau)\overset{\Delta}{=}\tau+d_{ik}(t+\tau)$ which is the combined waiting time and link delay for traversing link (i,k). To minimize traversal time we are interested in finding, for link (i,k) and time t, an optimal waiting time, i.e., a waiting time $\tau^*\geq 0$ such that for any other waiting time $\tau\geq 0$, $D_{ik}(t,\tau^*)\leq D_{ik}(t,\tau)$. In some cases such a value may not exist (see [6]). In the following we shall implicitly assume that delay functions are such that for all (i,k) and every time instant t a proper optimal waiting time τ^* does exist. This property clearly holds for continuous functions. With this in mind we define $D_{ik}(t)\overset{\Delta}{=}\min_{\tau\geq 0}\{D_{ik}(t,\tau)\}$, which is the combined link delay and optimal waiting time for traversing link (i,k) at time t.

Figure 1: Message departure and arrival times
(a) Immediate departures
(b) Arrival times for several departure times
(c) Optimal departure time

Denote by N_k the set of node k's neighbors; then, a *topological path* through the network is a sequence of nodes $(v_0, \ldots, v_m)$ such that a link exists between every pair (v_i, v_{i+1}) i.e., $v_{i+1} \in N_{v_i}$. These definitions of N_k and of a topological path are only topology dependent and not time dependent. Thus, a topological path corresponds to the ordinary graph-theoretic notion of path in the connection graph of the underlying network. A simple topological path is one in which no node appears more than once.

A *waiting schedule* is an m-tuple of waiting times $\boldsymbol{\tau}=(\tau_0,\tau_1, \ldots, \tau_{m-1})$ with $\tau_i \in [0,\infty)$ signifying the duration of waiting at node v_i. A *traversal path* is a an ordered pair $(\boldsymbol{\pi},\boldsymbol{\tau})$ of a topological path $\boldsymbol{\pi}$ and a waiting schedule $\boldsymbol{\tau}$, $\boldsymbol{\tau}$ having one less component than $\boldsymbol{\pi}$. A traversal path $(\boldsymbol{\pi},\boldsymbol{\tau})$ is called simple if $\boldsymbol{\pi}$ is a simple topological path. When no ambiguity exists, we shall use the term "path" to refer to both topological and traversal paths.

Let t_S be the earliest time in which a given message can start traversing the network (this is usually the message creation time). t_S is referred to as the *starting time*. Suppose that a given message travels from a source node s to its destination node w along some traversal path $(\boldsymbol{\pi},\boldsymbol{\tau})$, with $\boldsymbol{\pi}=(v_0,v_1, \ldots, v_m)$, $v_0=s$, $v_m=w$, $\boldsymbol{\tau}=(\tau_0,\tau_1, \ldots, \tau_{m-1})$. Define the arrival time t_A at a node in the following way:

$$t_A(0) \stackrel{\Delta}{=} t_S$$

$$t_A(i) \stackrel{\Delta}{=} t_A(i-1) + D_{v_{i-1},v_i}(t_A(i-1), \tau_{i-1}) \qquad 0 < i \le m$$

$$t_D(i) \stackrel{\Delta}{=} t_A(i) + \tau_i \qquad 0 \le i < m$$

For starting time t_S, $t_A(i)$ is the arrival time at node v_i and $t_D(i)$ the departure time from that node when traversing the network according to $(\boldsymbol{\pi},\boldsymbol{\tau})$. The *path delay* $PD((\boldsymbol{\pi},\boldsymbol{\tau}),t_S)$ of a path $(\boldsymbol{\pi},\boldsymbol{\tau})$ for starting time t_S is given by

$$PD((\boldsymbol{\pi},\boldsymbol{\tau}),t_S) = t_A(m) - t_A(0)$$

The shortest path problem for a graph $G(V,E,D)$ can now be formulated. Given nodes s and w in V and a time $t_S \in [0,\infty)$, find a traversal path $(\boldsymbol{\pi},\boldsymbol{\tau})$, $\boldsymbol{\pi}=(s,v_1, \ldots, v_{m-1},w)$, $\boldsymbol{\tau}=(\tau_0,\tau_1, \ldots, \tau_{m-1})$ such that for any other traversal path $(\boldsymbol{\pi}',\boldsymbol{\tau}')$ with $\boldsymbol{\pi}'=(s,u_1, \ldots, u_{l-1},w)$

$$PD((\boldsymbol{\pi},\boldsymbol{\tau}),t_S) \le PD((\boldsymbol{\pi}',\boldsymbol{\tau}'),t_S) \ .$$

Such a traversal path is a minimum delay path. We shall also refer to it as a "shortest path." The topological-path component of a minimum delay traversal path is referred to as a shortest topological path.

A shortest topological path may have the property that each of its subpaths is also a shortest topological path between the source and the intermediate nodes for the same starting time. Such a topological path is said to be *concatenated*. Formally, a shortest topological path $\boldsymbol{\pi}=(v_0,v_1, \ldots, v_m)$ between nodes v_0 and v_m for starting time t_S is said to be concatenated if for each i $0 \le i \le m-1$ there is a schedule $\boldsymbol{\tau}^{(i)}$ such that $((v_0,v_1, \ldots, v_i), \boldsymbol{\tau}^{(i)})$ is a shortest traversal path between v_0 and v_i for starting time t_S. Note that the schedule for each of the subpaths may be different. Consider now a shortest traversal path whose corresponding topological path is concatenated. If the optimal schedule of each subpath is the corresponding sub-schedule of the entire traversal schedule we say that the shortest traversal path is also concatenated. Formally, a shortest traversal path $(\boldsymbol{\pi},\boldsymbol{\tau})$, $\boldsymbol{\pi}=(v_0,v_1, \ldots, v_m)$, $\boldsymbol{\tau}=(\tau_0,\tau_1, \ldots, \tau_{m-1})$, for v_0, v_m and starting time t_S is said to be concatenated if for each i, $0 \le i \le m-1$, the traversal path $((v_0,v_1, \ldots, v_i), (\tau_0,\tau_1, \ldots, \tau_{i-1}))$ is a shortest traversal path for v_0, v_i, and t_S.

Finally, while all the shortest paths have the same delay they may differ by the number of hops (i.e., number of links). Considering those shortest paths which are simple and concatenated, and assuming that there are such paths, we define the hop-index $H(s,w,t)$ as the minimal number of hops among these simple and concatenated shortest paths.

B. Previous Results

In this subsection we summarize known results relating to shortest path algorithms for time-dependent networks. These results either indicate general properties or refer to centralized algorithms. They are the basis of the construction of the distributed protocols described in this paper. These results are mostly taken from Orda and Rom[6].

Given a source node s and a starting time t_S and allowing unrestricted waiting, we look for shortest paths and minimum delays between s and all other nodes for that starting time. As noted in Dreyfus[10] this is a straightforward extension to such algorithms as Dijkstra's[11] or Bellman-Ford's[12]. In other words, in the UW model, for any source-destination pair and any starting time, there exists a simple and concatenated shortest path. Therefore, there exist efficient polynomial algorithms that build a spanning tree of G rooted at s on which each path from s to any node j is a shortest path for starting time t_S whose delay is computed within the same algorithm.

Extending the above result for all times (rather than for a single given time) means dealing with time-dependent quantities. A main difficulty lies with the need to construct a spanning tree for each moment of time. To overcome this difficulty we use a set of functions $Y_{kl}(t)$ that is defined for every node k and for each of its neighbors $l \in N_k$. Let $X_l(t)$ be the earliest arrival time at node l for starting time t and let $Y_{kl}(t)$ similarly, be the earliest arrival time at node l through its neighboring node k. For a given time t and node k let l be a value such that $Y_{kl}(t)=X_l(t)$. Then, for each t the relation $l \rightarrow k$ defines a spanning tree of G rooted at s. Moreover, every path along the tree from s to every other node is a shortest path for starting time t. An algorithm to compute these can be specified[6].

In [6] it is indicated that when waiting is excluded at all nodes the problem becomes intractable. However, by just alleviating the constraint on waiting slightly, namely by permitting source waiting, efficient algorithms can be found. In particular if all the delay functions are continuous then there is equivalence (from the shortest path standpoint) between the UW and SW models in the sense that every shortest path in the UW model is also a shortest path in the SW model having the same delay. The proof of this property can be found in [6]. It should be noted that while the same path is being traversed, the departure times for each intermediate node may be different. Clearly, therefore, if all $d_{ik}(t)$ are continuous functions then, for a given source and destination pair and a starting time, there exists a shortest path in the SW model that is simple and concatenated.

As in the case of the UW model, for the SW model an efficient algorithm can be constructed, such that for a given starting time it computes waiting times and builds a spanning tree of G rooted at s to be interpreted as follows. If departure from s is delayed by the computed waiting then each path from s to any node j along the tree is a no-waiting shortest path for the given starting time.

C. The Delay Functions

We make the following assumptions on both $d_{ik}(t)$ and $D_{ik}(t)$ (for all $(i,k) \in E$):

A1. $d_{ik}(t)$ and $D_{ik}(t)$ are finite and continuous.

A2. $D_{ik}(t)$ is bounded, i.e.: there is a value $0<D_{\max}$ such that, for all i,k,t, $D_{ik}(t) \leq D_{\max}$.

A3. There is an integer valued finite function $F(L)$, such that for any finite time interval of length L, $d_{ik}(t)$ and $D_{ik}(t)$ can be described by at most $F(L)$ parameters. For example, if $d_{ik}(t)$ and $D_{ik}(t)$ are polynomials of degree n, then $F(L)=n+1$.

A4. The Bounded Variation Assumption (defined formally below).

A5. The Compactness Assumption (defined formally below).

D. Transmission Delays

The delay functions $d_{ik}(t)$ describe, in fact, the transmission delay of a *regular* (i.e., non-protocol) message, that have a fixed maximal size. However, in this paper we shall consider transmissions of messages whose length can be well beyond that of regular messages. Indeed, the distributed protocols presented in the following section consist of the exchange of control messages, which carry descriptions of functions of time whose length cannot be assumed to be fixed. We assume that all delay functions incorporate delays for regular messages, and can therefore treat regular messages as being of zero length. We assume that a message of excess size b i.e., that contains b "data-units" (parameters) beyond the size of a regular message, which is transmitted at time t on link (i,k), arrives to node k at time $d_{ik}(t)+\alpha_{ik}\cdot b$, where α_{ik} is a finite coefficient of link (i,k). In other words, the head of the message arrives at time $d_{ik}(t)$, whereas each subsequent data-unit beyond the regular message size arrives α_{ik} time units afterwards. This corresponds to the frozen link delay model, in which the time-variance of the link delay applies to the message head.

E. The Distributed Environment

Throughout the paper we follow the terminology of [13] and refer to the program residing in each node as the *node algorithm* and to the ensemble of all algorithms as the *distributed protocol*.

We assume that each node knows the identity of its neighbors, but has no knowledge of the topology of the rest of the network. Node i is cognizant of the functions $d_{ik}(t)$ and $D_{ik}(t)$ for each of its neighbors k, but does not have any knowledge regarding the functions of any other link (except for implicit knowledge communicated by the protocols).

Although the functions $D_{ik}(t)$ are upper-bounded by $D_{\max}$, this bound may be unreasonably high for adopting a synchronous mode of distributed operation. Thus, we shall deal essentially with an asynchronous environment for distributed computation. We also assume that the only factor that affects the execution time of a distributed protocol is link delay. Processing times can be included in link delays and are therefore ignored. In particular, for long messages, the coefficient α_{ik} can account also for the extra processing time per unit of length.

We assume that whenever a message is sent from a node i to a neighbor k, it arrives correctly and in finite time to node k; finiteness results immediately from the finiteness of the link delay functions and of the coefficient α_{ik}. Node k is cognizant of the link through which a message arrives.

As mentioned above, a protocol control message may be lengthy and thus its reception at a node may take some time, beyond that of regular messages. This may delay the processing of the data contained in that message. However, within the time of a regular message the node can clearly process the header of the arriving message and thus identify it as a protocol message and react promptly. In other words, within the allocated time a node can react to the arrival of the message but not the data contained in it (beyond the header). In the following, we shall use the term *upon receiving the header of (message) X* when referring to the time at which an incoming control message X is recognized as such (i.e., its header has been received and processed). Also, we shall use the term *upon completing the reception of (message) X* when referring to the time by which an incoming control message X has been *fully* received and processed. When referring to *message complexity,* we mean the number of control messages used by a given protocol, regardless of their length.

Nodes are required to perform operations on functions. The reader is referred to [6] for a discussion on such operations. For our purposes it suffices to state that nodes are capable of performing the following operations on the functions considered: function assignment ($f(t) \leftarrow g(t)$), linear combination of two functions ($c_1 g_1(t) + c_2 g_2(t)$), minimum of two functions ($\min\{g_1(t), g_2(t)\}$), compounding two functions ($f(g(t))$).

We shall deal mainly with "tree-based" protocols. We assume that a directed spanning tree is defined in the network, rooted at a selected node s, such that every node $i \neq s$ knows the identity of its parent p_i on that tree. Denote by R the diameter of that tree. We assume that our protocols are invoked by higher level protocols that create "START" impetuses at node s.

We are now ready to present our last two assumptions regarding delay functions. To that end, we need the following terminology. For a source node i and a destination node s, consider paths between i and s with at most n hops, where $0 \leq n \leq |V|-1$. Among such paths, denote by $STP(i,s,t,n)$ the identity of the shortest topological path for source node i, destination node s and starting time t. If no such path exists then $STP(i,s,t,n)$ is taken to be the (unique) empty path. Also, since there may be multiple such paths, we break ties in the following way: the waiting time at each node is chosen to be maximal (while still keeping the path be a shortest one), and ties among neighbors are broken by node ID. $STP(i,s,t,n)$ is, therefore, unique. We note that the above rules of waiting time maximization and tie-breaking are applied throughout the paper for identifying unique shortest paths. Consider now an interval of time I of length L. Denote by $S(i,I,n)$ the set of times at which node i changes its (n-hops restricted) shortest-path $STP(i,s,t,n)$, i.e.: $S(i,I,n) = \{t \mid t \in I,\ STP(i,s,t^-,n) \neq STP(i,s,t,n)\}$.

Bounded Variation assumption (A4): For a selected (destination) node s, there is a finite integer valued function $G(L)$ such that, for any finite time interval I of length L, for all nodes i and for all n, $0 \leq n \leq |V|-1$:

$$|S(i,I,n)| \leq G(L)\ .$$

The meaning of the above assumption is that shortest-paths do not change too frequently. In the next section we discuss a practical interpretation of this assumption.

Our last assumption is rather technical and is concerned with the relation between the length of time needed for transmitting the description of a delay function and the length of the time interval that the transmitted function describes. The following assumption simply states that the (excess) time needed for transmitting the description of a function defined on some interval is not large with respect to the interval's length.

Compactness Assumption (A5): There is some $\gamma>0$, known to the destination node s, small with respect to $D_{\max}$, such that

$$F(|V|\cdot(2\cdot R\cdot D_{\max}+\gamma))\cdot G(|V|\cdot(2\cdot R\cdot D_{\max}+\gamma))\leq\frac{\gamma}{(|V|-1)\cdot\max\limits_{(i,k)\in E}\alpha_{ik}}$$

The above may be interpreted in the following way. Since R is the diameter of a spanning tree and $D_{\max}$ is an upper bound on a link traversal, $2\cdot R\cdot D_{\max}$ is an upper bound on a round-trip delay. This could be considered as a network epoch. The product of the functions $F(\cdot)$ and $G(\cdot)$ is related (in a sense that will be made precise later) to the amount of information needed for describing shortest-paths during $|V|$ time epochs on a link. The value of $\max\limits_{(i,k)\in E}\alpha_{ik}$ relates the amount of information to its transmission time. Thus, the Compactness Assumption states that within a period of duration γ one can pass the information regarding shortest-paths in $|V|$ epochs on any link. Note that γ appears also in the argument of $F(\cdot)$ and $G(\cdot)$, since an "epoch" should also include the time needed to describe it (this shall become clear in the sequel). We point out that, throughout the paper, the assumption regarding the upper bound $D_{\max}$ is used only in the context of the Compactness Assumption. Furthermore, the values of $D_{\max}$, R and α_{ik} need not be known to any node. The selected node s should be cognizant of the values of $|V|$ and γ.

III. DISTRIBUTED SHORTEST PATH PROTOCOLS

We present in this section distributed protocols for finding the shortest path and minimum delay in networks with continuous time dependent link delays for the UW model. In the end of the section we show how to adapt these protocols to the SW model. These protocols compute the shortest paths and delays from all nodes to a given destination node, where distributed computation of a path amounts to determining for every node the next node along that path and the respective departure time. More precisely, the protocols compute functions that enable each node i to calculate, for each moment of time, the minimal delay to the destination s through each neighbor. With these functions in hand, a node can determine the proper routing of a message immediately at any point of time, by choosing the neighbor that minimizes the delay.

These protocols are based on known ones[13,14] that are adapted to deal with time varying link delays. A main issue that must be dealt with in the adaptation is that, although each node knows the link delays for all times on all its outgoing links, exchanging such functions is a nontrivial operation. We solve this problem by limiting the interval for which shortest paths are computed. This means that the functions to be exchanged must be valid only within a limited interval for which a finite parametric representation will do. This will enable us to keep the messages at a manageable length.

To introduce these concepts we first present protocols that compute the shortest path for all $t \geq \tilde{t}$ for a sufficiently large $\tilde{t}$ so we can assume that protocol execution terminates prior to $\tilde{t}$; then we present protocols that operate on finite-length intervals. We point out that an infinite-interval protocol may necessitate dealing with functions whose representation is infinite since assumption A3 applies only to finite intervals.

We search for shortest paths from all nodes in the network to a given node s (referred to as the destination). This is a switch from the computation done in [6] but is typical in network routing problems where nodes need to route their messages to the destination node and prefer doing it along the shortest path. In a network with fixed delays each node will start such a protocol once, and after termination every node will know the shortest path to the destination node. In our case, since we wish to limit ourselves to intervals (for compact function representation) the protocol runs continuously for every node.

The protocols work by exchanging *control messages* of the form $MSG(ID, delay, value)$ where ID is the sender's ID or number, $delay$ is some delay function as will be explained later, and $value$ is a numerical value appearing in some protocols. Control messages are exchanged among neighbors only. As a consequence, both in the UW and SW models control messages are allowed to wait before departing on the link. Thus, a control message flowing from node i to node k can be regarded as though it traverses through a link whose delay function is given by $D_{ik}(t)$, implying that the FIFO property holds for control messages.

Since the protocol runs for a given destination s, and since many of these protocols run concurrently, one ought to carry the ID of s on every message; in our specification, however, we omit this notation.

In the description of the protocols we refer to several characteristic moments in the lifetime of messages (see Section II). The time at which a message is created is called its starting time, the time at which a message arrives at some intermediate node $i \neq s$ is called the arrival time, and the time at which a message arrives at the destination node s is called the final arrival time.

A. Protocols for $t \geq \tilde{t}$

1. Fixed Tree Protocol

We begin with an informal presentation of DSPP1--the first distributed shortest path protocol we present. DSPP1 runs on a fixed, predetermined spanning tree of the network rooted at s.

The centralized algorithms presented in [6] are based on Dijkstra's and Bellman-Ford's algorithms and thus their distributed implementation through a synchronizer, such as *Synchronizer* α of [15], should be straightforward. Nonetheless, we present a formal specification and verification of DSPP1 for two reasons. First, this serves as a first step towards the finite intervals version presented in the sequel. Second, we point out that the algorithms of [6] are for a common source whereas our protocols work for a common destination. This calls for an explicit verification of the "reversed" common destination version.

DSPP1 consists of an initialization step followed by $|V|-1$ iterations, after which it achieves its goals. Each iteration begins at node s by sending a control message to all its neighbors; the first iteration begins when s receives an external "START" impetus, and subsequent ones whenever node s completes an iteration and determines that another one is necessary. Receiving replies from all its neighbors indicates to node s that an iteration ended. Every other node (all but node s) begins an iteration upon receiving (the header of) a control message from its respective parent, upon which it sends control messages to all its neighbors but the parent. When replies are (fully) received from all these neighbors a control message is sent to the parent thereby terminating the iteration at that node.

The control messages contain the minimal delay function (known thus far during the execution) between that node and node s for all starting times. When such a message is fully received at node i from its neighbor l, node i checks whether the new information improves its minimum delay function to node s. It does so by considering paths that begin with the edge (i,l), taking into account the most recent information just reported by node l.

As explained earlier (section II.E) a node reacts to an incoming control message in two stages: first, as soon as the header is received and the message is identified as a control one, the node may send control messages to its neighbors (according to the protocol); then, after the entire message is received, the node updates its variables according to the contents of the message. A possible problem that may arise is that node s may start a new iteration before all nodes have finished receiving the control messages of the previous iteration. This indeed happens if the values of the delay functions (which determine the transmission times of headers) are short compared to the delay incurred by control messages of excess size. For reasons of clarity, at this stage we simply require that node s wait for a period of Γ before beginning a new iteration, where Γ is assumed to be long enough to guarantee the complete reception of all messages of this iteration. This issue will be addressed in detail in the next subsection when dealing with the finite-interval scheme.

We proceed with a formal specification of the protocol. Node i maintains the following as part of the protocol:

- $M_i(l)$ -- a boolean variable indicating whether node i has received a message from neighbor l in the current iteration.
- $\Delta_i(l,t)$ -- the delay from node i to node s on a shortest path among those starting with neighbor l for a message either arriving or starting at time t at node i.
- $\delta_i(t)$ -- the minimum delay from node i to node s for a message either arriving or starting at node i at time t.

Node i is assumed to know the functions $D_{ij}(t)$ that are the delay functions that correspond to the maximum optimal waiting time (see Section II for their definition). In addition node i knows p_i--its parent on the tree.

Protocol DSPP1

Algorithm for node s.

Initialization: $count \leftarrow 0 \;\; \forall l \in N_s \quad M_s(l) \leftarrow 0$.

S1. Upon Receiving START
   a. send $MSG(s, 0)$ to all $j \in N_s$.

S2. Upon receiving the header of $MSG(l, \delta_l(t))$
   a. $M_s(l) \leftarrow 1$.
   b. If $\forall j \quad M_s(j)=1$ then
      A. $\forall j \quad M_s(j) \leftarrow 0$
      B. $count \leftarrow count+1$
      C. Wait for Γ units of time.
      D. i. If $count = |V|-1$ then Stop.
         ii. Else send $MSG(s,0)$ to all $j \in N_s$.

Algorithm for node $i \neq s$.

Initialization: $\forall l \in N_i \quad M_i(l) \leftarrow 0 \;\; \Delta_i(l,t) \leftarrow \infty \;\; \delta_i(t) \leftarrow \infty$.

1. Upon receiving the header of $MSG(l, \delta_l(t))$
   a. $M_i(l) \leftarrow 1$.
   b. If $l = p_i$ send $MSG(i, \delta_i(t))$ to all $j \in N_i$ except p_i.
   c. If $\forall j \quad M_i(j)=1$ then
      i. Send $MSG(i, \delta_i(t))$ to p_i.
      ii. $\forall i \quad M_i(j) \leftarrow 0$.
2. Upon completing the reception of $MSG(l, \delta_l(t))$
   a. $\Delta_i(l,t) \leftarrow \min\{\Delta_i(l,t), \delta_l(t+D_{il}(t))+D_{il}(t)\}$
   b. $\delta_i(t) \leftarrow \min_{j \in N_i}\{\Delta_i(j,t)\}$

Note that $\Delta_i(l,t)$ and $\delta_i(t)$ are not reset from iteration to iteration meaning that in each iteration the previously computed functions are updated.

For the purpose of stating and proving the next theorem we introduce the following notation and definition. For each time t, consider all those neighbors $j \in N_i$ of node i having the minimal $\Delta_i(j,t)$ i.e., $\forall k \in N_i$, $\Delta_i(j,t) \le \Delta_i(k,t)$. Among all these we denote by $P_i(t)$ the *preferred neighbor*, i.e., the one for which the waiting time at node i for arrival time t is maximal (if there is more than one such node, we choose arbitrarily the one with least ID number). We note that, for time t and neighbor k, we can compute the maximal waiting time at node i that enables arriving at node k on link (i,k) as early as possible by finding a maximal τ in the interval $[0, D_{ik}(t)]$ such that $\tau + d_{ik}(t+\tau) = D_{ik}(t)$. By definition $P_s(t) \overset{\Delta}{=} NIL$. Let $PATH(i,t)$ be a path obtained by starting at node i at time t and moving along according to the value of $P_i(\cdot)$. Formally, for each node i and all times $t \ge \tilde{t}$ $PATH(i,t) \overset{\Delta}{=} (i_0 = i, i_1, i_2, \ldots)$, $i_{k+1} = P_{i_k}(t_k)$, $t_{k+1} = t_k + D_{i_k, i_{k+1}}(t_k)$, $t_0 = t$. Since for all $i \ne s, t$: $P_i(t) \ne NIL$, PATH must either be infinite or terminate at s.

Theorem 1:

1. Suppose node s receives a "START". Then it will perform step <S2.b.D.i> in finite time and exactly once. When this happens all nodes $i \ne s$ will have completed the algorithm, and exactly one message will have traversed each link in each direction in each iteration.

2. After termination $\forall i, t$ $PATH(i,t)$ is a simple and concatenated (with respect to the destination) shortest topological path for i, s, t whose delay is $\delta_i(t)$.

Proof: Part 1 follows directly from Segall[13].

From our definition it follows that $PATH(i,t)$ is composed of $PATH(j, t + D_{ij}(t))$ appended to node i where $j = P_i(t)$. To prove that $PATH(i,t)$ is a simple path we first introduce the following lemma.

Due to the assumption made on Γ, we know that, by the time a new iteration begins, all nodes have received and processed the messages of the previous iteration, i.e., have performed step <2> for all incoming messages. Thus, at each iteration n, the function $\delta_i(t)$ sent by a node at step <1> of this (nth) iteration, is the function computed according to the functions $\Delta_i(l,t)$ received by i during the $(n-1)$st iteration.

Lemma 1: $\forall n \ge 1 \; \forall i$ and for $t_2 > t_1$ $t_2 + \delta_i(t_2)_n \ge t_1 + \delta_i(t_1)_n$. (The notation VAR_n denotes the value of VAR after the nth iteration of the protocol, i.e., after the corresponding node executed step <2> during the nth iteration; omitting the subscript refers to all iterations).

Proof: Assume the claim of the lemma to be false, i.e., $\exists\, n \ge 1$, i, and for some $t_2 > t_1$: $t_2 + \delta_i(t_2)_n < t_1 + \delta_i(t_1)_n$. At the beginning of execution (initial state) the claim was true; thus we may consider the first moment during protocol execution (as seen by an external observer) in which the claim was first violated. At that time let n, i, t_1, t_2, be respectively the iteration number, node, and times. Consequently $t_2 + \delta_i(t_2)_n < t_1 + \delta_i(t_1)_n$ and thus by step <2> there exists a node l, an iteration $1 \le m \le n$ and a previous time during the execution for which

$$t_2 + D_{il}(t_2) + \delta_l(t_2 + D_{il}(t_2))_m < t_1 + \delta_i(t_1)_n \le t_1 + \delta_i(t_1)_m \le t_1 + D_{il}(t_1) + \delta_l(t_1 + D_{il}(t_1))_m$$

Defining $t_1' \overset{\Delta}{=} t_1 + D_{il}(t_1)$ and $t_2' \overset{\Delta}{=} t_2 + D_{il}(t_2)$ the last inequality can be written as

$$t_2' + \delta_l(t_2')_m < t_1' + \delta_l(t_1')_m \quad .$$

From the definition of $D_{il}(t)$ it follows that $t_2 > t_1$ implies $t_2^{'} \geq t_1^{'}$. Looking at the above inequality we observe a node l, an iteration $m \leq n$ and a pair of times $t_1^{'}$ and $t_2^{'}$ for which the claim was violated at a previous moment of protocol execution, contradicting the assumption. □

We now proceed to prove that $PATH(i,t)$ is simple. Assume the contrary, i.e., there exists a node j, a time t_0, and an iteration n such that j appears more than once in $PATH(j,t_0)_n$ ($PATH(j,t_0)_n$ is defined using $P_{i_k}(t_k)_n$ instead of $P_{i_k}(t_k)$ where $P_{i_k}(t_k)_n$ is constructed from $\Delta_{i_k}(l,t_k)_n$). Directly from the definition and for some $m \leq n$ we have

$$t_0 + \delta_j(t_0)_n = t_0 + \Delta_j(P_j(t_0)_n, t_0)_m = t_0 + D_{jP_j(t_0)_n}(t_0) + \delta_{P_j(t_0)_n}(t_0 + D_{jP_j(t_0)_n}(t_0))_m$$
$$\geq t_0 + D_{jP_j(t_0)_n}(t_0) + \delta_{P_j(t_0)_n}(t_0 + D_{jP_j(t_0)_n}(t_0))_n \equiv t_0^{'} + \delta_{P_j(t_0)_n}(t_0^{'})_n$$

where the step before last follows from the observation that $\delta_j(t)_{k+1} \leq \delta_j(t)_k$.

The above, applied repeatedly in the presumed loop will eventually lead to $t_0 + \delta_j(t_0)_n \geq t_1 + \delta_j(t_1)_n$ for some $t_1 > t_0$. Strict inequality is ruled out by the previous lemma and thus equality must hold namely $t_0 + \delta_j(t_0)_n = t_1 + \delta_j(t_1)_n$. Since $P_j(t)$ is chosen to maximize waiting time it follows that $P_j(t_0)_n = P_j(t_1)_n$. Repeating this argument implies the existence of an infinite loop, i.e., the existence of an infinite sequence of times $t_0 < t_1 < t_2 < \cdots < t_k < \cdots$ for which $t_0 + \delta_j(t_0)_n = t_k + \delta_j(t_k)_n$ for all k. From the protocol it is easy to see that $\delta_j(t)_n$ is finite for $n \geq 1$. $t_{k \to \infty} \to \infty$ because $d_{ik}(t)$ is continuous and strictly positive and thus has strictly positive lower bound in any finite interval. Since t_0 is finite we are lead to a contradiction and $PATH(i,t)$ is therefore a simple path.

Since $PATH(i,t)$ is simple it will terminate at s. Also, from the protocol it follows that $\delta_i(t)$ is the delay along this path from node i to node s for starting time t. What is left to show is that $PATH(i,t)$ is indeed a shortest path, which we do next.

We show that, for $m \geq 0$, $\delta_i(t)_m$ is upper-bounded by the minimum delay for i,s,t among paths with at most m hops (if no such path exists, then $\delta_i(t)_m = \infty$). The proof is by induction on m; the claim is trivially true for $m=0$, and assuming true for $m-1$ we prove for m.

Consider a specific time t_0 and a node i such that there is a path with at most m hops between i and s. Let $SP(i,s,t_0,m)$ be a shortest traversal path with at most m hops between i and s for starting time t_0, Let k be i's successor on $SP(i,s,t_0,m)$ and let t_1 be the corresponding arrival time to k on that path. This means that there is a path with at most $m-1$ hops between k and s. By the inductive assumption, for all times t, $\delta_k(t)_{m-1}$ is the minimum delay on paths with at most $m-1$ hops between k and s and starting time t. In the m-th iteration, node k sends to node i this function, and in particular the value of $\delta_k(t_1)_{m-1}$, thus guaranteeing that $\delta_i(t_0)_m$ is upper-bounded by the delay of $SP(i,s,t_0,m)$, thus completing the inductive step. Let $n = H(i,s,t_0)$. The above implies that, for $m \geq n$, $\delta_i(t_0)_m$ is the minimum delay for i, s, t_0. Since t_0 was chosen arbitrarily, this holds for all values of t.

After $|V|-1$ iterations $\delta_i(t)$ will therefore be the minimum delay for all nodes $i \in V$.

Since we showed that $PATH(i,t)$ is a simple path whose delay is $\delta_i(t)$ we conclude that it is a simple shortest topological path. From its construction it is evident that $PATH(i,t)$ is concatenated. □

It should be noted that $|V|-1$ is an upper bound for the number of iterations needed. One could terminate the protocol after the first iteration in which $\delta_i(t)$ did not change for any of the nodes. We conclude from the above theorem that the message complexity of DSPP1 is $O(|V|\cdot|E|)$.

Having executed DSPP1, routing is performed as follows. Upon receipt of a message at node i (either from the external world or from a neighbor) at time t and destined to node s, node i routes it to a neighbor k for which $D_i(k,t)=\delta_i(t)$.

2. A Fast, No-Tree Protocol

DSPP1, being executed on a fixed tree may exhibit poor performance due to poor tree selection. We briefly discuss here a second protocol, DSPP2, that is also performed iteratively, but its iterations propagate in the quickest possible way [16]. This protocol is of a rather common type[14,13]. In fact, DSPP2 corresponds to a local synchronizer such as *Synchronizer* β of [15] for which it is known that a faster execution results. This discussion is warranted because the straightforward implementation cannot be incorporated within the finite intervals approach.

As before, the first iteration of DSPP2 begins with a "START" at node s causing it to "awaken" its neighbors and proceed with protocol execution. From this point on each node performs its iterations autonomously without a trigger from s, which plays the role of a regular node. Upon the receipt of the first message (in the first iteration) a node sends its own first message. A node begins a new iteration (beyond the first) after having received a message belonging to the current iteration from each of its neighbors.

To manage the various iterations going on concurrently, iteration numbers must be maintained. It turns out that at most two successive iterations can go on simultaneously between two neighbors; hence it is necessary to maintain only two sets of indicators corresponding to $M_i(l)$ in DSPP1, and a one bit iteration counter. For details see [16].

B. Protocols for Finite Intervals

We have seen that DSPP1 terminates, i.e., a finite time after invocation at node s every node i knows its minimum delay and the next hop in a shortest path towards s for all $t \geq t'$. This leads to the conclusion that there exist times $\tau_1, \tau_2, \cdots$ such that if the protocol is started at node s at time τ_{i-1} it will terminate before τ_i. The idea is to design a protocol that runs during such finite intervals, in every execution exchanges information relating to another future finite interval, and terminates in time for this information to be used. This section deals with the determination of these intervals and their evaluation by distributed protocols. But first we elaborate further on the interval concept.

We define an interval-protocol for node s and times t_1^s, t_2^s as a protocol upon whose termination all nodes will have computed the minimum delays and shortest paths for messages with final arrivals at node s during $[t_1^s, t_2^s]$. For continuous delay functions such a protocol defines for every node i an interval $[t_1^i, t_2^i]$ such that either starting or arriving at node i during this interval and proceeding along the shortest path will result in final arrival at node s during $[t_1^s, t_2^s]$. Moreover, because the FIFO property holds between source and destination in the UW and SW models when the shortest path is used, in these models there exists a monotonic nondecreasing function mapping starting or arrival times in $[t_1^i, t_2^i]$ to final arrival times $[t_1^s, t_2^s]$.

Our view of operation is a repeated execution of the protocol so that in spite of it addressing a limited interval per iteration, every node will know at time t its minimum delay and shortest path for that time.

1. Fixed Tree Protocol

Our interval protocol, DISPP1, is an adaptation of DSPP1 and runs on a predetermined spanning tree. The protocol runs in consecutive intervals $[t_0^s{=}0, t_1^s]$, $[t_1^s, t_2^s]$,..., $[t_n^s, t_{n+1}^s]$ ... During the n-th interval $[t_n^s, t_{n+1}^s]$, DISPP1 performs two tasks: it runs DSPP1 (to completion) for the interval $[t_{n+1}^s, t_{n+2}^s]$ and it determines t_{n+3}^s. Since calculations performed by DISPP1 are for $[t_{n+1}^s, t_{n+2}^s]$ only, we shall prove (c.f., Corollary 1), that functions sent by node i to its neighbors need be valid only within the finite interval $[t_{n+1}^i, t_{n+2}^s]$. In the specification of the protocol we thus implicitly assume that $\delta_i(t)$ and $\Delta_i(l,t)$ are defined for that interval only.

Consider a typical interval $[t_n^s, t_{n+1}^s]$. To determine t_{n+3}^s a termination-time protocol is used. This protocol operates exactly like the DSPP1 except that instead of exchanging the delay *functions* $\delta_i(t)$ it exchanges the *times* at which every node will receive the DSPP1 messages during the interval starting at t_{n+2}^s. When it completes t_{n+3}^s is determined. A trivial upper-bound on the termination time could be obtained by relying on the value of $D_{\max}$. However, this could result in an excessively large interval. The alternative method presented in the following results in a more efficient (smaller) value, since it does not rely on $D_{\max}$ except through the use of the Compactness Assumption.

Assume first that protocol messages are of regular size. To compute t_{n+3}^s every node i maintains a variable q_i whose value is the latest arrival time of a DSPP1 message in the interval $[t_{n+2}^s, t_{n+3}^s]$. Hence, when the protocol terminates the value of q_i is the time at which the last DSPP1 message for that interval will have arrived at node i. The same applies to node s where upon termination $q_s = t_{n+3}^s$, given that it started with the value of t_{n+2}^s. Thus the value of t_{n+2}^s is given to s as part of the START impetus, and when node s finishes the execution of its algorithm for the current interval, it returns (to some higher-level control algorithm) the value of t_{n+3}^s.

The above corresponds to a control message of a regular size. However, the actual messages may be longer, and thus the calculation of the interval length must be modified. What is needed is to guarantee that node s does not begin a new iteration before all other nodes have completed the reception of control messages belonging to the previous iteration. This is accomplished simply by imposing on s an additional delay of γ time units before completing each iteration, where γ is the value given in the Compactness Assumption. We now show that when the additional delay ends, all nodes have indeed completed the reception of previous messages. We derive first a bound on the length of the interval. This corresponds to the execution time of the protocol, which consists of $|V|-1$ iterations. The execution time of each iteration is bounded by $2 \cdot R \cdot D_{\max} + \gamma$, in which $2RD_{\max}$ is the longest round-trip time along the spanning tree (recall that R is its diameter), and γ is derived from the Compactness Assumption. Next we derive a bound on the length of a control message, i.e., of a function $\delta_i(t)$. By the proof of Theorem 1, at the n-th iteration this function describes the delays of shortest-paths among paths with at most $n-1$ hops. By Assumption

A4, during an interval of length L the identity of such paths (for a given source node and iteration number) changes at most $G(L)$ times. Each such path consists of at most $|V|-1$ links, thus its delay function for an interval of length L can be described, due to assumption A3, by at most $(|V|-1)\cdot F(L)$ parameters. Putting all the above together, the (excess) transmission time of a control message on any link is at most

$$\max_{(i,k)\in E} \alpha_{ik}\cdot(|V|-1)\cdot F((|V|-1)\cdot(2\cdot R\cdot D_{\max}+\gamma))\cdot G((|V|-1)\cdot(2\cdot R\cdot D_{\max}+\gamma))\leq\gamma$$

where the inequality follows from the Compactness Assumption (A5). Thus, by allowing an extra delay of γ units before proceeding, node s makes sure that all previous messages completely arrive.

We now present a formal specification of DISPP1, which includes the distributed computation of t^s_{n+3}. Because it simulates DSPP1 operation, determining termination time requires the same number of iterations as is required by DSPP1 itself (bounded by $|V|-1$). Protocol DISPP1 therefore consists of exchanging messages merging the information contained in DSPP1 messages with the information needed to determine termination time. Because both require the same number of iterations and the same number of messages per iteration this merger is done on a message by message basis. The following is a specification of the protocol execution during interval n.

Protocol DISPP1

Algorithm for node s.

Initialization: $count\leftarrow 0$ $\forall l\in N_s$ $M_s(l)\leftarrow 0$.

S1. Upon Receiving START with $\tau=t^s_{n+2}$
   a. $q_s\leftarrow\tau$
   b. Send $MSG(s, 0, q_s+D_{sj}(q_s))$ to every $j\in N_s$.

S2. Upon receiving the header of $MSG(l, \delta_l(t), q)$
   a. $M_s(l)\leftarrow 1$.
   b. If $q>q_s$ then $q_s\leftarrow q$.
   c. If $\forall j$ $M_s(j)=1$ then
      A. $\forall j$ $M_s(j)\leftarrow 0$
      B. $count\leftarrow count+1$
      C. Wait for γ units of time.
      D. $q_s\leftarrow q_s+\gamma$
      E. If $count=|V|-1$ then
         i. $count\leftarrow 0$.
         ii. Return $t^s_{n+3}=q_s$
         Else
         i. Send $MSG(s, 0, q_s+D_{sj}(q_s))$ to every $j\in N_s$.

Algorithm for node $i\neq s$.

Initialization: $q_i\leftarrow 0$, $\delta_i(t)\leftarrow\infty$, $M_i(l)\leftarrow 0$, $\forall l\in N_i$: $\Delta_i(l,t)\leftarrow\infty$.

1. Upon receiving the header of $MSG(l, \delta_l(t), q)$

a. $M_i(l) \leftarrow 1$.
b. If $l = p_i$ then send $MSG(i, \delta_i(t), q_i + D_{ij}(q_i))$ to every $j \in N_i$ except p_i.
c. If $\forall j$ $M_i(j) = 1$ then
   i. Send $MSG(i, \delta_i(t), q_i + D_{ip_i}(q_i))$ to p_i.
   ii. $\forall i$ $M_i(j) \leftarrow 0$.

2. Upon completing the reception of $MSG(l, \delta_l(t), q)$
   a. If $q > q_i$ then $q_i \leftarrow q$.
   b. $\Delta_i(l, t) \leftarrow \min\{\Delta_i(l, t), \delta_l(t + D_{il}(t)) + D_{il}(t)\}$
   c. $\delta_i(t) \leftarrow \min_{j \in N_i}\{\Delta_i(j, t)\}$

The properties of DISPP1 are all derived from the following theorem.

Theorem 2: Repeated execution of DISPP1 (i.e., invoking a new execution of the protocol at node s as soon as the previous one ends) guarantees that $\forall t$ every node i will have computed the shortest path $SP(i, s, t)$ and minimum delay $\delta_i(t)$ prior to time t, assuming proper values for t_1^s and t_2^s.

Proof: Assume the existence of times t_n^s, t_{n+1}^s, and t_{n+2}^s such that execution of DISPP1 starting at node s at times t_n^s and t_{n+1}^s will terminate no later than t_{n+1}^s and t_{n+2}^s respectively, and assume the corresponding times at node i to be t_{n+1}^i and t_{n+2}^i. To prove the theorem it is sufficient to show that if DISPP1 is started at node s at time t_n^s then, upon termination:

1. Every node i knows the minimum delay and shortest path from it to node s for final arrival times at node s in the interval $[t_{n+1}^s, t_{n+2}^s]$.
2. q_i is the time at which the iteration that starts at node s at time t_{n+2}^s terminates at node i.
3. $\forall i$ $t_{n+2}^i \le q_i \le t_{n+3}^i$.

It is straightforward to recognize that removing the computations relating to q_i (i.e., steps <S1.a>, <S2.b>, <S2.c.D>, <S2.c.E.ii> and <2.a>) and setting $\Gamma = \gamma$ converts DISPP1 to DSPP1. In particular, note that from the above discussion it follows that γ of DISPP1 complies with the assumption made on Γ of DSPP1. Since execution starting at t_n^s will terminate prior to t_{n+1}^s we may set $\tilde{t} = t_{n+1}^s$ and thus part 1 above results directly from Theorem 1.

Following the values of q exchanged (steps <S1.b>, <S2.c.D>, <1.b>, <1.c.i>) it is clear that the q values indeed simulate message arrival times (note that the extra delay γ is identical in all intervals). Since the value of q_i is updated to be maximal (steps <S2.b> and <2.a>), part 2 above follows directly.

Node s that starts the protocol sends its neighbors q values larger than t_{n+2}^s but since, by necessity, $t_{n+2}^i < t_{n+2}^s$ we clearly have $q_i \ge t_{n+2}^i$. By the nature of the protocol node i completes every iteration before node s does, thus for a message to leave node i at some time q_i and finally arrive at node s before t_{n+3}^s it must leave node i not after t_{n+3}^i. □

Since execution starting at node s at time t_n^s terminates no later than t_{n+1}^s, we conclude that the functions $\delta_i(t)$ that are contained in the messages sent by node i, need be valid only within $[t_{n+1}^i, t_{n+2}^i]$.

Unfortunately, node i does not know the value of t_{n+2}^i at time t_n^i and the problem is to determine an interval whose bounds are known. We have seen that when starting the nth interval node s knows the value of t_{n+2}^{s} and we shall require that this value be appended to the first message sent by each node to each neighbor during this interval. Since node i knows at the beginning of the n-th interval the minimum delays for final arrival times in $[t_n^s, t_{n+1}^s]$, it can determine the value of t_{n+1}^{i} which is the latest time t for which $t+\delta_i(t)=t_{n+1}^{s}$. Thus, before it starts sending messages during the n-th interval node i already knows the values of t_{n+1}^{i} and t_{n+2}^{s}, and since $t_{n+2}^{s}>t_{n+2}^{i}$ we conclude that node i may send during the n-th interval functions that are valid only within $[t_{n+1}^{i}, t_{n+2}^{s}]$. Thus we have the following corollary:

Corollary 1: The functions $\delta_i(t)$ exchanged in the messages need be valid only within $[t_{n+1}^{i}, t_{n+2}^{s}]$.

□

The above theorem and corollary capture the essence of DISPP1. If DISPP1 is started at node s at time t_n^s (for $n \geq 1$) it will terminate prior to t_{n+1}^{s} having carried the shortest path information for the interval $[t_{n+1}^{i}, t_{n+2}^{i}]$. Having also determined t_{n+3}^{i} this procedure is repeated indefinitely. Note that the message complexity of DISPP1, per execution, is identical to that of DSPP1, that is: $O(|V| \cdot |E|)$.

The results obtained in Theorem 2 and corollary 1 are depicted in Figure 2. The figure describes the occurrence of events over time in node s and another typical node i. The dotted lines denote relations between arrival (or starting) times at node i and the corresponding optimal final arrival times at node s. We focus on the n-th interval.

At time t_n^s, when node s begins execution of the n-th interval, node i knows the optimal delay for messages destined to node s whose arrival or starting times are up to t_{n+1}^{i} (which fulfills $t_{n+1}^{i}>t_n^s$). At time t_n^s protocol control messages are sent from node s and the first control message arrives at node i at some

Figure 2: Characteristic protocol operation in the n-th interval

time denoted in the figure by b_i. Node i enters the current interval at that time, leaving it at some later time q_i. When this happens, node i knows the minimal delay for its $n+1$st interval of starting times i.e., up to time t^i_{n+2}. Since Theorem 2 guarantees that $q_i \le t^i_{n+1}$, node i is always ready to route messages to s optimally.

At time q_i node i sends to its parent p_i the last control message of this interval (step <1.c.i>); later on, p_i does the same to its parent, until at some time Q_i node s receives a message from some ancestor of node i in the tree. Since t^s_{n+1} was chosen to be the time at which the iteration terminates at node s, $Q_i \le t^s_{n+1}$. Also note that $\hat{q}_i$ which is the corresponding latest starting time at node i for which Q_i is the optimal final arrival time, must fulfill $\hat{q}_i \ge q_i$; this is the basis for the claim (in Theorem 2) that $q_i \le t^i_{n+1}$. (Note that the values b_i, Q_i, and $\hat{q}_i$ have been introduced for explanation purposes only and do not appear in the protocol).

Finally, observe that, according to Corollary 1, the functions $\delta_i(t)$ sent by node i to its neighbors during $[b_i, q_i]$ need be valid only within $[t^i_{n+1}, t^s_{n+2}]$ (in fact as was mentioned earlier they could have been defined for $[t^i_{n+1}, t^i_{n+2}]$ if node i were able to know t^i_{n+2} at time b_i).

The theorem states that to make all this work two ‘proper’ initial values for t^s_1 and t^s_2 must be determined (the first two intervals may be considered the transient period of the operation). Proper, in this sense, means that these two values must be large enough to allow DISPP1 to complete the first and second interval computation on time. For practical cases it is possible to find such values. For example, for zero-length control messages, we can assume that at time $t=0$ node s is cognizant of an upper-bound $\hat{D}_{\max}$ on delay values such that, for $0 \le t \le 4 \cdot R \cdot \hat{D}_{\max}$ and for all $(i,k) \in E$ it holds that $d_{ik}(t) \le \hat{D}_{\max}$. (We note that $\hat{D}_{\max}$ may be smaller than $D_{\max}$.) This means that each of the first two intervals lasts at most $2 \cdot R \cdot \hat{D}_{\max}$ time units. An extension for dealing with nonzero length control messages is straightforward.

2. A Fast, No-Tree Protocol

It is impossible to extend DSPP2 to operate on bounded intervals in the same manner as was DSPP1 since to do so one must simulate future iterations of DSPP2 to determine the termination time of each DSPP2 invocation. In other words, we must first find out how each DSPP2 iteration propagates--a computation requiring itself $O(|V|)$ iterations and making the entire task impossible. However, the following extension of DSPP2 enables constant updating and a finite time domain environment (although the exact bounds of the domain cannot be predetermined).

Assume that when node s performs a "Stop" (in DSPP1), an interval computation is complete and the computation for the next interval may start. From the properties of DSPP2 it is easily concluded that no more than two intervals will be computed in the network at any time, lending itself to interval computation. Nonetheless, the nodes cannot compute the exact boundaries of these intervals (as is done in DISPP1), and thus some, usually inefficient, upper bound must be used, e.g.: $2RD_{\max}$. Note that node s is not necessarily the one that must begin a new interval; any node could do it after performing its "Stop". However, it may be advantageous, sometimes, to exert a bit of central control, for example in order to ensure that intervals do not become too short resulting in an excessive amount of message exchange.

C. Distributed Source Waiting-Time Computation

When the SW model is being used every node must compute the optimal source waiting time functions $WAIT(i,s,t)$, in addition to the shortest path and preferred neighbor. Although this function is noncontinuous we propose here a method in which the waiting time is computed by exchanging only continuous functions.

Corresponding to the functions $\Delta_i(l,t)$ and $\delta_i(t)$ of DSPP1 (and in addition to them) we define two new functions $\hat{\Delta}_i(l,t)$ and $\hat{\delta}_i(t)$ that play similar roles but for traveling along some simple path without waiting at intermediate nodes.

The protocol itself is identical to DSPP1 except that it uses messages of the form $MSG(ID,\delta(\cdot),\hat{\delta}(\cdot))$, for which the following modifications of DSPP1 are made:

- Initialize all $\hat{\Delta}_i(l,t)$ and $\hat{\delta}_i(t)$ to ∞.
- Add step 2.d: $\hat{\Delta}_i(l,t) \leftarrow \min\{\hat{\Delta}_i(l,t), \hat{\delta}_l(t+d_{il}(t))+d_{il}(t)\}$
- Add step 2.e: $\hat{\delta}_i(t) \leftarrow \min\limits_{j \in N_i}\{\hat{\Delta}_i(j,t)\}$

These steps are identical to steps <2.b> and <2.c> except that $d_{il}(t)$ is used instead of $D_{il}(t)$ (that is, excluding any waiting). Thus, if waiting is not necessary we expect $\hat{\delta}_i(t)$ to equal $\delta_i(t)$. The difference between the two are the basis for determining waiting times.

We now derive the basic property of $\hat{\delta}_i(t)$ relating to waiting time. Assume that in the SW model we always choose the latest departing time from the source that still achieves minimum delay. For each node i we can therefore divide the set of starting times $t \geq \tilde{t}$ into two sets: a nonempty set T_{I_i} of starting times for which departure is immediate (i.e., zero waiting), and a (possibly empty) set T_{D_i} of starting times for which departure is delayed (i.e., nonzero waiting). These two sets are exhaustive and mutually exclusive. With regard to these we can state the following:

Lemma 2: After execution of the protocol we have $\forall i \; \forall t_0 \geq \tilde{t}$:

if $t_0 \in T_{I_i}$ then $\hat{\delta}_i(t_0) = \delta_i(t_0)$

if $t_0 \in T_{D_i}$ then $\hat{\delta}_i(t_0) \geq \delta_i(t_0)$.

Proof: $\hat{\delta}_i(t_0)$ is the delay from node i to s for time t_0 without enroute waitings and is, by definition, no less than the minimal SW delay. By the property of the SW model (see Section II.B) the latter equals the minimum UW delay which, by Theorem 1, is $\delta_i(t)$. In other words $\hat{\delta}_i(t_0) \geq \delta_i(t_0)$ and thus only the first statement of the lemma is left to be proven.

Consider a simple and concatenated shortest path in the SW model for some given starting time $t_0 \in T_{I_i}$. Let this path be $(i_0, i_1, \ldots, i_m)$ with $i = i_0$ and $s = i_m$, and denote by t_{D_k} the departure time from node i_k. This path is also a simple and concatenated shortest path for the UW model for starting time t_0 (by the above mentioned property) but in which traveling is *without waiting* (since the departure time belongs to T_I). From the above observations and from the calculation step (<2>) it is evident that after protocol execution $\hat{\delta}_{i_k}(t_{D_k}) = \delta_{i_k}(t_{D_k})$ leading to $\hat{\delta}_i(t_0) = \delta_i(t_0)$. □

Theorem 3: Define $WAIT(i,t)$ as that maximal $\tau \geq 0$ for which the expression $\tau + \hat{\delta}_i(t+\tau)$ is minimized, and let $PATH(i,t)$ be as defined in Theorem 1. Then $PATH(i, t_S + WAIT(i, t_S))$ is a shortest path in the SW model for source node i, starting time t_S, and source waiting time $WAIT(i, t_S)$.

Proof: From lemma 2 it follows that after protocol execution $\hat{\delta}_i(t)$ is an "over estimate" of $\delta_i(t_0)$ for $t_0 \in T_{D_i}$ and is identical for $t_0 \in T_{I_i}$ meaning that one should wait until that time in which the two functions are identical. The above definition of waiting time ensures, in addition, maximum waiting time at the source. Since messages will be leaving node i only at times when $\hat{\delta}_i(t) = \delta_i(t)$ the theorem follows directly from Theorem 1. □

The above theorem makes our approach clear. We first run the distributed protocol described above and obtain for each $i \in V$ the function $\hat{\delta}_i(t)$. Then, upon the receipt of a data message destined to node s at source node i and starting time t, node i calculates the corresponding source waiting time according to Theorem 3.

It should be noted that the maximality requirement in the definition of the *WAIT* function results from our definition of $P_i(t_0)$. Note also that since the calculation of waiting time is done using $\hat{\delta}_i(t)$ alone and since the delay itself can be derived from $\hat{\delta}_i(t)$ all the calculation and exchange of information regarding $\Delta_i(l,t)$ and $\delta_i(t)$ is actually unnecessary but was included to simplify the presentation and the proof.

The extension of the above protocol for fast protocols and for finite intervals is straightforward. Note also that the above protocol has the same message complexity as DSPP1, that is: $O(|V| \cdot |E|)$.

In fact, we could use the above protocol also in the UW model, as an alternative for DSPP1 (and the corresponding DISPP1). Indeed, according to [6], the minimal delay in the SW model is identical to that of the more general UW model. A possible advantage of the above protocol to DSPP1 is that it does not require to compute the values of $D_{ik}(t)$.

D. Model Relaxations

In this subsection we indicate how our results can be accommodated to handle more relaxed models. We begin by showing how one can cope with uncertainty.

Till now we have been considering a deterministic model, in which there is full knowledge regarding the future evolvement of the network. However, in many practical cases the dynamic behavior of the network may not be predicted in a precise way; rather, one has some good estimate regarding the possible values of link delays in the nearby future. In general, finding optimal paths in a stochastic time-dependent network is an extremely difficult problem (see [17]). However, it is possible to adapt the intervals scheme to an environment that involves some degree of uncertainty.

Assume that for each link (i,k) two functions are given. The first is an estimate of the delays on that link in future times and the second is an upper-bound on possible values of delays in future times. It is assumed that traveling along paths that minimize the estimated delays render good performance (although delays are not guaranteed to be minimal, due to the uncertainty involved). We incorporate these functions in DISPP1 in the following way. The phase that computes shortest paths shall use the estimated functions (step <2.b> of the algorithm), whereas the phase that simulates a future execution uses the upper-bounded functions (steps <S1.b>, <S2.c.D>, <2.a> of the algorithm).

During the simulation phase at which t^s_{n+1} is determined (i.e., during the $n-2$nd interval) node i calculates the simulated time at which it will finish the execution for that interval. Recall that this value is obtained through upper bounds on delays. At the end of the $n-2$nd interval t^s_{n+1} is determined. To this value corresponds a starting time $\tilde{t}^i_{n+1}$ at node i, which is related to t^s_{n+1} through the *upper-bounded* functions. Using the same reasoning as in Section III, we conclude that the simulated finishing time will be before node i actually needs it.

During the actual protocol execution at the nth interval, control messages flow faster than what was simulated (since in the simulation phase we used upper-bounds) meaning that the actual times at which node i actually enters and leaves the protocol are before those computed during the simulation. In particular, it follows from the conclusion of the previous paragraph that the actual time at which node i leaves the protocol is earlier than $\tilde{t}^i_{n+1}$. The starting time t^i_{n+1}, related to t^s_{n+1} through the *estimated* functions, fulfills $\tilde{t}^i_{n+1} \le t^i_{n+1}$, since the estimated functions cannot supersede the upper-bounds. This means that the actual time at which node i leaves the protocol is not later than t^i_{n+1}. On the other hand, it is easy to verify that the actual entering time is still later than t^i_n. Thus, we finally conclude that in this case too the protocol execution for the nth interval at node i is confined to within the corresponding nth interval of that node. Thus, the protocol execution at a node is still locked within the corresponding interval of starting times, meaning that the protocol still has the important property that a node always has the relevant information at all times. The only deficiency is that the lengths of these intervals may be larger than what is optimally achievable with full knowledge.

Other possible relaxations are with respect to the assumptions made on the delay functions. In the following we outline a relaxation of the Bounded Variation assumption. The idea is that, instead of making the Bounded Variation be an *assumption* on the functions, one could view it as a *constraint* on the way shortest-paths can be chosen, since it is clear that switching between paths too often is impractical. Thus, a node would be allowed to switch between shortest-paths at some maximal rate of r, i.e., $G(L) \le r \cdot L$. This might introduce errors if the true functions change more frequently. However, these errors can be bounded. For example, if link delay functions change at a rate less than ρ then it can be shown that the error is bounded by $2(|V|-1)\rho/r$. Note that the error goes to 0 with ρ/r, i.e., when functions are smooth enough.

E. Comparison with Other Approaches

In this subsection we compare our approach with others. We first consider a centralized approach as an alternative. Just as with regular, time-independent networks, distributed approaches may be substituted by the straightforward centralized approach of sending the entire information to a central node that computes the optimal solutions and disseminates them to all nodes. Indeed, such an approach has its advantages, e.g., data consistency. It should be noticed that a centralized approach would still necessitate an intervals scheme for which the intervals scheme of DISPP1 can be used.

We opted for the distributed approach since it has several important advantages compared with the centralized one. One major advantage is balancing the load among the nodes in terms of the required protocol processing which is of particular importance when complex operations on functions are considered. In addition, the distributed approach tends to reduce the load of control messages in the area near the central node. In order to realize this, consider the following example. Let the destination node s be the central node in the centralized protocol, and assume that it is connected to the rest of the network by a single link (s,u). We compare the load on that link in each of the two protocols assuming the two protocols use the same intervals. Consider first our protocol (DISPP1). In each iteration, a control message traverses (s,u) in each direction: in the direction $s \rightarrow u$, the message carries the zero-value function, whereas in the direction $u \rightarrow s$, the message carries the link-delay function $d_{us}(t)$. In both cases, these messages are "short", compared to regular minimum-delay functions $\delta_i(t)$. Thus, during the execution of DISPP1, $|V|-1$ short messages flow in each direction of (s,u). Consider now the centralized protocol. In the first phase, the central node must get the delay functions of all links in the network, meaning that $|E|-1$ short messages traverse (u,s). In the second phase, s sends the minimum delay function $\delta_i(t)$ to each node i. Each of these functions corresponds to a path having as many as $O(|V|)$ hops, making a message carrying such a function a "long" message (note that a "long" message can be $\Omega(|V|)$ times longer than a short one). This means that, during an execution of the centralized protocol, on link (s,u) as many as $O(|E|)$ short messages traverse in the direction $u \rightarrow s$ and $O(|V|)$ long messages traverse in the direction $s \rightarrow u$, making the advantage of DISPP1 obvious.

Yet another naive approach is to drop shortest path computations altogether and use instead a flooding scheme, which is usually a very simple way of guaranteeing delivery of each message as fast as possible. Such an approach has obvious disadvantages, most notably an extremely high flow of data messages, and the need to keep track of message identities. These disadvantages are particularly relevant to time-dependent networks (and dynamic networks in general), where messages may wander around for a long time before disappearing, due to variability in link delays. Moreover, a flooding scheme is invalid in the SW model, since in that model initiating the flooding by the source node at an arbitrary time would usually result in a non-minimal delay. This means that the source node should figure out at which time the flooding should be initiated, thus requiring again a shortest-path computation. As an example (which meets assumptions A1--A5), consider a three-node network, with $V=\{1,2,3\}$ and $E=\{(1,2),(2,3)\}$. The delay functions are:

$$d_{12}(t)=\varepsilon \qquad d_{23}(t)=\begin{cases} D_{\max} & 0\le t<\varepsilon \\ D_{\max}\cdot(2-\frac{t}{\varepsilon}) & \varepsilon\le t<\varepsilon\cdot(2-\frac{1}{D_{\max}}) \\ \varepsilon & \varepsilon\cdot(2-\frac{1}{D_{\max}})\le t \end{cases}$$

where $D_{\max}\gg 1$ and $\varepsilon \ll 1$. For starting time 0, the minimal delay between nodes 1 and 3 is $\varepsilon\cdot(3-\frac{1}{D_{\max}})$, achieved (in the SW model) by leaving node 1 at time $\varepsilon\cdot(1-\frac{1}{D_{\max}})$. However, a flooding scheme, initiated at time 0, results in a delay as high as D_{max}+ε.

IV. CONCLUSION

In this paper we have shown how basic distributed shortest path protocols can be adapted to work in a time-dependent environment. This adaptation calls for special care in two respects. First, the analysis of these protocols (even the straightforward one, DSPP1) is more subtle due to the time-varying nature of the problem. Secondly, the need to exchange functions between nodes calls for a special solution, namely the intervals operation. We see the main contribution of the intervals scheme in the way it proposes to solve distributed problems involving functions on an infinite domain - by "slicing" them into finite domains, while obeying severe timing constraints.

Although we concentrated on a deterministic model, in which there is complete and precise knowledge regarding future dynamics, we indicated how our results can be adapted to environments that involve uncertainty. We also indicated that assumptions made on the delay functions can be relaxed.

As a last point, we note that the approach devised in this paper calls for elaborate exploitation of the information found in the network, in order to guarantee that the selected paths are indeed of minimum delay. This approach suits well cases in which one cares less on how much it takes to compute a path, as long as its delay is minimal and the information on it is available by the time it is to be used. A case that falls in this category is routing of messages with sensitive information.

REFERENCES

1. A. Segall, ‘‘The Modeling of Adaptive Routing in Data-Communication Networks,’’ *IEEE Trans. on Communications* **COM-25**(1) pp. 85-95 (January 1977).
2. E.J. Anderson, P. Nash, and A.B. Philpott, ‘‘A Class of Continuous Network Flow Problems,’’ *Mathematics of Operations Research* **7**(4) pp. 501-514 (November 1982).
3. G.I. Stassinopoulos and P. Konstantopoulos, ‘‘Optimal Congestion Control in Single Destination Networks,’’ *IEEE Trans. on Communications* **COM-33**(8) pp. 792-800 (August 1985).
4. G. Sasaki and B. Hajek, ‘‘Optimal Dynamic Routing in a Single Commodity Network by Iterative Methods,’’ *IEEE Trans. on Communications* **COM-35**(1) pp. 1199-1206 (November 1987).
5. A.B. Philpott, ‘‘Continuous-Time Flows in Networks,’’ *Mathematics of Operations Research* **15**(4) pp. 640-661 (November 1990).
6. A. Orda and R. Rom, ‘‘Shortest-Path and Minimum-Delay Algorithms in Networks with Time-Dependent Edge-Length,’’ *Journal of the ACM* **37**(3) pp. 607-625 (July 1990).
7. A. Orda and R. Rom, ‘‘Minimum Weight Paths in Time-Dependent Networks,’’ *Networks* **21**(3) pp. 295-319 (May 1991).
8. A. Orda and R. Rom, ‘‘Location of Central Nodes in Time Varying Computer Networks,’’ *Operations Research Letters* **10**(3) pp. 143-152 (April 1991).
9. N. Deo and C.Y. Pang, ‘‘Shortest Path Algorithms: Taxonomy and Annotation,’’ *Networks* **14** pp. 275-323 (1984).
10. S.E. Dreyfus, ‘‘An Appraisal of some Shortest Path Algorithms,’’ *Operations Research* **17** pp. 395-412 (1969).
11. E.W. Dijkstra, ‘‘A Note on Two Problems in Connection with Graphs,’’ *Numerical Analysis* **1** pp. 269-271 (October 1959).
12. L.R. Ford Jr. and D.R. Fulkerson, *Flows in Networks,* Princeton University Press, Princeton, New Jersey (1962).
13. A. Segall, ‘‘Distributed Network Protocols,’’ *IEEE Trans. on Information Theory,* pp. 23-34 (January 1983).
14. J.M. McQuillan, G. Falk, and I. Richer, ‘‘A Review of the Development and Performance of the ARPANET Routing Algorithm,’’ *IEEE Trans. on Communications* **COM-26**(12) pp. 1802-1811 (December 1978).

15. B. Awerbuch, ‘‘Complexity of Network Synchronization,’’ *Journal of the ACM* **32**(4) pp. 804-823 (October 1985).

16. Ariel Orda, ‘‘Time-Dependent Networks,’’ D.Sc. Dissertation, Department of Electrical Engineering, Technion, Haifa, Israel (1991).

17. A. Orda, R. Rom, and M. Sidi, ‘‘Minimum Delay Routing in Stochastic Networks,’’ *IEEE/ACM Transaction on Networking* **1**(2) pp. 187-198 (April 1993).