Stepwise Assertional Design of Distance–Vector Routing Algorithms*

Cengiz Alaettinoğlu, A. Udaya Shankar

Institute for Advanced Computer Studies
and Department of Computer Science
University of Maryland
College Park, Maryland 20742
ca@isi.edu, shankar@cs.umd.edu

UMIACS-TR-92-39, CS-TR-2869

August 1993

Abstract

There are many kinds of distance–vector algorithms for adaptive routing in wide–area computer networks, ranging from the classical Distributed Bellman–Ford to several recent algorithms that have better performance. However, these algorithms have very complicated behaviors and their analyses in the literature has been incomplete (and operational). In this paper, we present a stepwise assertional design of a recently proposed distance–vector algorithm. Our design starts with the Distributed Bellman–Ford and goes through two intermediate algorithms. The properties established for each algorithm hold for the succeeding algorithms. The algorithms analyzed here are representative of various internetwork routing protocols.

* This work is supported in part by National Science Foundation Grant No. NCR 89-04590, and by RADC and DARPA under contract F30602-90-C-0010 to UMIACS at the University of Maryland. The views, opinions, and/or findings contained in this report are those of the author(s) and should not be interpreted as representing the official policies, either expressed or implied, of the Defense Advanced Research Projects Agency, RADC, or the U.S. Government.

Contents

1 Introduction

Adaptive routing protocols are responsible for choosing optimal routes for data packets in wide-area store-and-forward computer networks such as the Internet. In these networks, each link has a cost (indicating the current traffic on the link) that changes with time; furthermore, links can fail and recover. We refer to such changes as **topology changes**. A routing protocol must monitor these topology changes and adapt its routes accordingly.

In a routing protocol, each node maintains for each destination a neighboring node id, referred to as its **next–hop**. The node forwards data packets destined for the destination to its next–hop. The next–hop can be *nil*, in which case the node does not know where to forward data packets for that destination. The objective can be informally stated as follows: (a) the succession of next–hops for the destination from any node should lead to the destination (unless the destination is unreachable); and (b) the cost of this next–hop path should be minimum amongst all paths from the node to the destination.

A popular approach to routing is the **distance–vector** approach, which is based on the Bellman-Ford algorithm [5]. In this approach, each node maintains for each destination a set of **distances**, one for each of its neighbors, and chooses a neighbor with minimum distance as the next–hop. Thus, each node requires $O(N \times e)$ space, where N is the number of nodes in the network and e is the average degree of a node. However it is well known that the straight-forward distributed implementation of the Bellman-Ford algorithm can have **long–lived loops** (of the order of distances) [14]. In fact, the ARPANET initially used this Distributed Bellman–Ford algorithm, but because of long–lived loops, it was replaced in 1979 by a brute–force "link state" algorithm which requires $O(N^2)$ space at each node (to maintain a view of the network topology with a cost for each link).

Since 1979, many new kinds of distance–vector algorithms have been proposed [15, 19, 10, 21, 3, 6, 17, 9] which avoid long–lived loops by using various node coordination mechanisms. For example [15, 19, 10] use diffusion computations [4] to avoid loops entirely. References [21, 3, 17, 9] avoid long–lived loops, but not **short–lived loops**, i.e. loops that disappear in time proportional to N or less. In [21], each node maintains for each destination a set of **paths** (in addition to the distances), one for each of its neighbors. The intention is that the path maintained at node u for a neighbor is the next–hop path of the neighbor with node u appended to the front. Long–lived loops are avoided by not choosing a neighbor as a next–hop if the path maintained for that neighbor contains a loop. However maintaining and exchanging paths is expensive and requires $O(N \times e \times H)$ storage at each node, where H is the length (in number of links) of a maximum length shortest cost path between any two nodes (note that H can be as high as N). References [3, 17, 9] overcome this problem by having nodes maintain **prefinal nodes** instead of the paths. The prefinal node for a destination is intended to be the last node before the destination on the next–hop path. Using the prefinal nodes, a node can reconstruct the path to any destination (see Section 6), thereby avoiding long–lived loops.

Understanding distance–vector algorithms, particularly the new ones, is difficult. The analyses in the literature of the above algorithms (e.g. showing that optimal paths are eventually achieved) are operational and incomplete. In this paper, we present a stepwise assertional design of distance–vector algorithms. We go through the following steps:

(1) We start our design with the Distributed Bellman–Ford algorithm, referred to as **A1**. We prove that after any succession of topology changes, the nodes that can still reach the destination eventually achieve and maintain optimal next–hop paths.

(2) We next obtain an algorithm, called **A2**, by adding a path–exchange mechanism to **A1**. We prove that **A2** converges to optimal paths in $O(N)$ steps, assuming **synchronous execution** of the network; i.e. the routing algorithm executes in steps, and in each step all (and only those) messages that are send in the previous step are received. This proves that **A2** avoids long–lived loops.

(3) We next obtain an algorithm, called **A3**, by adding to **A2** a constraint that a node chooses a neighbor as the next–hop for a destination only if the neighbor is also the next–hop for all intermediate destinations on the path to the destination.

(4) Our fourth algorithm, called **A4**, is obtained from **A3** by replacing paths with prefinal nodes.

For each algorithm **A**i, the safety and progress properties satisfied by the previous algorithms hold. In the case of **A2** and **A3**, it is straightforward to check that the proofs for the previous algorithms continue to hold with minor modifications. For **A4**, we establish that **A4** is a **well–formed refinement** [11] of **A3**; thus, all safety and progress assertions satisfied by **A3** hold for **A4** [11].

Many algorithms proposed in the literature use similar mechanisms to algorithms **A1** through **A4**. For example, Old Arpanet Routing Algorithm [14], Routing Information Protocol (an Internet standard) [7], and Inter-Gateway Routing Protocol [8] are variations of **A1**. Inter-Domain Routing Protocol (ISO draft standard) [18], Border Gateway Protocol (an Internet standard) [12], and the algorithm in [21] are variations of **A2**. The algorithms in [3, 17, 9] are variations of **A4**. Hence, understanding the properties of algorithms **A1** through **A4** is very useful in understanding various internetworking routing protocols. We introduce **A3** because showing that properties of **A2** hold for **A4** is not simple (whereas showing that properties of **A2** hold for **A3**, and properties of **A3** hold for **A4** is simple).

In section 2, we present our system model and proof rules. In sections 3, 4, 5, 6, we describe **A1**, **A2**,

A3, and **A4**, respectively. In section 7, we give concluding remarks. A preliminary version of algorithms **A1** through **A4**, without most of the analysis, was presented in [1].

2 Preliminaries: System Model and Proof Rules

We use state transition systems and fairness requirements to specify routing protocols, and safety and progress assertions to describe their behaviors (e.g. [11, 20, 13]).

A state transition system consists of a set of state variables, a set of events, and an initial condition on the state variables. The state variables define the set of system states. Each event e is specified by an enabling condition, referred to as $enabled(e)$ and an (atomically executed) action, referred to as $action(e)$; together they define a set of state transitions for the event.

A *behavior* of the state transition system is a sequence of the form $\langle s_0, f_0, s_1, f_1, \ldots \rangle$, where the s_i's are system states, the f_i's are event names, s_0 is an initial state, and for each $i \geq 0$, (s_i, s_{i+1}) is a transition of f_i. A behavior can be infinite or finite (in which case it ends in a state). In the following definitions, we consider behavior $\sigma = \langle s_0, f_0, s_1, f_1, \ldots \rangle$.

An event can be subject to a *weak fairness*. A behavior σ satisfies weak fairness for event e iff (1) σ is finite and e is not enabled in the last state of σ, or (2) σ is infinite and either e occurs infinitely often or is disabled infinitely often in σ.

We use two types of safety assertions in this paper: invariant assertions and unless assertions. An invariant assertion is of the form $Invariant(A)$ where A is a state formula, i.e. a formula which is true or false at each state. By definition, $Invariant(A)$ holds for a behavior σ iff every state s_i in σ satisfies A.

An unless assertion is of the form A *unless* $B \vee \mathcal{E}$, where A and B are state formulas and $\mathcal{E}$ is a set of event names. By definition, A *unless* $B \vee \mathcal{E}$ holds for a behavior σ iff for every state s_i in σ satisfying $A \wedge \neg B$, at least one of the following hold: (1) s_i is the last state (σ is finite), or (2) s_{i+1} satisfies $A \vee B$, or (3) f_i is in $\mathcal{E}$. The event set $\mathcal{E}$ can be empty, in which case we simply write A *unless* B.

A safety assertion holds for a state transition system iff it holds for every behavior of the system.

Our progress assertions are of the form A *leads–to* $B \vee \mathcal{E}$, where A and B are state formulas and $\mathcal{E}$ is a set of event names. By definition, A *leads–to* $B \vee \mathcal{E}$ holds for a behavior σ iff for every s_i in σ that satisfies A, there is a $j \geq i$ such that s_j is in σ and satisfies B or f_j is in σ and belongs to $\mathcal{E}$. The event set $\mathcal{E}$ can be empty, in which case we simply write A *leads–to* B. Given a state transition system and a set of fairness requirements, a leads–to assertion holds for the system iff it holds for every behavior of the system which satisfies the fairness requirements.

We next list the proof rules used in this paper. We use $Initial$ as a state formula specifying the initial condition. Given an event e, we use $\{A\}e\{B\}$ to mean the Hoare–triple $\{A \wedge enabled(e)\}action(e)\{B\}$, i.e., in any state that satisfies A, if e is enabled then its occurrence results in a state that satisfies B.

Invariance rule: *Invariant*(A) holds if for some state formula C, the following hold:

- $Initial \Rightarrow A$
- for every event e, $\{A \wedge C\}e\{A\}$
- *Invariant*(C).

Implication rule: *Invariant*(A) holds if for some state formula C, the following hold:

- *Invariant*(C)
- $C \Rightarrow A$.

Unless rule: A *unless* $B \vee \mathcal{E}$ holds if for some state formula C, the following hold:

- for every event $e \notin \mathcal{E}$, $\{A \wedge \neg B \wedge C\}e\{A \vee B\}$
- *Invariant*(C).

Leads–to rule: A *leads–to* $B \vee \mathcal{E}$ holds if for some state formula C, the following hold:

- for every event $e \notin \mathcal{E}$, $\{A \wedge \neg B \wedge C\}e\{A \vee B\}$
- for some event e with weak fairness, $\{A \wedge \neg B \wedge C\}e\{B\}$
- *Invariant*($A \wedge C \Rightarrow enabled(e)$)
- *Invariant*(C).

Closure rules:

- A *leads–to* $B \vee \mathcal{E}$ holds if *Invariant*($A \Rightarrow B$) holds.
- A *leads–to* $B \vee \mathcal{E}$ holds if for some state formula C: A *leads–to* $C \vee \mathcal{E}$ and C *leads–to* $B \vee \mathcal{E}$ hold.
- A *leads–to* $B \vee \mathcal{E}$ holds if $A = A_1 \vee A_2$, A_1 *leads–to* $B \vee \mathcal{E}$, and A_2 *leads–to* $B \vee \mathcal{E}$ hold.
- $A \wedge B$ *leads–to* $(C \vee (A \wedge D)) \vee \mathcal{E}$ holds if A *unless* $C \vee \mathcal{E}$ and B *leads–to* $D \vee \mathcal{E}$ hold.

These rules are similar to the rules in [13, 2]. It is straightforward to show their soundness (e.g. [11, 20]).

3 Algorithm A1

We consider a computer network whose nodes and links form an arbitrary directed graph such that if there is a link from node u to node v, then there is a link from node v to node u. Let *NODES* be the set of nodes, and *LINKS* ($\subseteq$ *NODES* $\times$ *NODES*) be the set of links. Node v is a neighbor of node u if (u, v) is a link. Let

$neighbors(u)$ denote the set of neighbors of u. A sequence $\langle x_0, \ldots, x_n \rangle$ of nodes is a *path* iff (x_i, x_{i+1}) is a link for $0 \leq i < n$. A path is *simple* if no node is repeated.

A routing protocol is specified by a state transition system and a set of fairness requirements. Each node u has a set of state variables and a set of events. Each link (u, v) has a state variable, called $Channel_{uv}$, indicating the sequence of messages in transit. $Channel_{uv}$ initially equals $\langle\rangle$, the null sequence. The events of a node can access the state variables of the node, send messages to outgoing links, and receive messages from incoming links. A link (u, v) behaves as a FIFO queue, except when it fails, in which case $Channel_{uv}$ is set to $\langle\rangle$; for notational convenience, we group this failure event among the events of node u. We assume that each receive event has weak fairness; this is a convenient way to model finite message propagation delays.

Conventions: We use u, v, w, x, y, z to range over $NODES$; in some (explicitly stated) cases, they range over $NODES \cup \{nil\}$. We use v, w to range over $neighbors(u)$. We use z to indicate the destination node. We use $c, k, d, newcost$ to range over $I^+ \cup \{0, \infty\}$, indicating a distance or a cost, where I^+ is the set of positive integers. We treat ∞ as a number higher than any number in I^+; e.g. ∞ plus any number is ∞. Given a set S of numbers, $\min S$ denotes the smallest number in S. If S is empty then $\min S$ returns ∞.

Table 1 specifies the state variables and events of an arbitrary node in **A1**, the Distributed Bellman–Ford algorithm. (Refer to the table in the following discussion.) Node u maintains the cost of each outgoing link (u, v) in state variable $Linkcost_u(v)$. $Linkcost_u(v)$ equals ∞ iff the link is failed; it can change its value at any time due to link failure, link recovery and link cost change events. $Linkcost_u(v)$ is never 0.

For each destination z, node u maintains in state variable $Distvia_u(v, z)$ an estimate of the distance to z via neighbor v. It equals ∞ if node u believes z cannot be reached via v. The state variable $Nhop_u(z)$ indicates the next–hop for destination z. It equals neighbor v only if $Distvia_u(v, z)$ is minimum among all neighbors. $Nhop_u(z)$ equals nil iff $Distvia_u(v, z)$ equals ∞ for all neighbors v. Node u also maintains state variable $Dist_u(z)$ in which it stores the distance via its next–hop, except when $u = z$ (in which case $Dist_z(z) = 0$).

Nodes exchange information about their distances to various destinations. Specifically, node v sends messages of the form (v, d_vector), where d_vector is a set of (z, d) pairs such that $d = Dist_v(z)$; note that d can be ∞.

When $Linkcost_u(v)$ changes (either because of link failure, recovery or change in cost), $Distvia_u(v, z)$, and if needed $Nhop_u(z)$ and $Dist_u(z)$, is updated for each destination z (for details see procedure $Update\&Send$ in table 1). If the distance of any destination z has been affected (i.e. $Dist_u(z)$ has changed), node u sends a message to its neighbors containing the $(z, Dist_u(z))$ pairs for all affected destinations z.

Additionally, when link (u, v) recovers, u sends a message to v containing the $(z, Dist_u(z))$ pairs for all destinations z. This is to ensure that if u offers a better path for some destination z, node v will choose u as its next–hop. This also ensures that if a network become connected after being disconnected (due to a set of link failures), nodes in different partitions obtain paths to each other.

When node u receives a (v, d_vector) message, it updates $Distvia_u(v, z)$, and if needed $Nhop_u(z)$ and $Dist_u(z)$, for each destination z in d_vector. If the distance of any destination has been affected, node u sends a message to its neighbors containing the $(z, Dist_u(z))$ pairs for all affected destinations z.

We say that the network is in a *symmetric* state if for every link (u, v), link (u, v) is up iff link (v, u) is up. In the rest of this section, we prove that after any succession of topology changes that leaves the network symmetric, for every node u and every destination z reachable from u, eventually the next–hop path starting from u leads to z and has minimum cost among all paths from u to z. To specify this formally, we define the following functions (on the system state):

UPLINKS. Set of up links. Formally,

$= \{(u, v) \in \mathit{LINKS} : Linkcost_u(v) < \infty\}$.

Symmetric. Boolean.

$= \mathit{true}$ iff $[\forall (u, v) \in \mathit{LINKS} : (u, v) \in \mathit{UPLINKS}$ iff $(v, u) \in \mathit{UPLINKS}]$.

Nhoppath(u, z). The succession of next–hops for z starting from u. Formally,

$= \langle x_0, \ldots, x_n \rangle$ such that $x_0 = u$,

for $i \in [0..n-1] : Nhop_{x_i}(z) = x_{i+1},\ \wedge\ x_i \notin \{x_0, \ldots, x_{i-1}\} \cup \{nil\} \cup \{z\}$, and

$x_n = z\ \vee\ Nhop_{x_n}(z) = nil\ \vee\ x_n \in \{x_0, \ldots, x_{n-1}\}$.

Availablepaths(u, z). The simple paths from u to z over up links. Formally,

$= \{\langle x_0, \ldots, x_n \rangle : x_0 = u\ \wedge\ x_n = z\ \wedge\ [\text{for } i \in [1..n] : (x_{i-1}, x_i) \in \mathit{UPLINKS}\ \wedge\ x_i \notin \{x_0, \ldots, x_{i-1}\}]\ \}$.

Reachable. Set of node pairs (u, z) such that u can reach z. Formally,

$= \{(u, z) : Availablepaths(u, z) \neq \{\}\ \}$.

Path_cost$(\langle x_0, \ldots, x_n \rangle)$. The cost of path $\langle x_0, \ldots, x_n \rangle$. Formally,

$$= \begin{cases} \sum_{i=0}^{n-1} Linkcost_{x_i}(x_{i+1}) & n > 0 \\ 0 & n = 0 \quad \text{(i.e. path equals } \langle x_0 \rangle\text{)} \\ \infty & n < 0 \quad \text{(i.e. path equals } \langle\rangle\text{)} \end{cases}$$

Note that the path cost is ∞ if any link cost in the path is ∞.

Cost(u, z). The cost of a minimum cost path from u to z. Formally,

$= \min\{Path\_cost(p) : p \in Availablepaths(u, z)\}$.

$HighestCost = \max\{Cost(u,z) : (u,z) \in Reachable\}$.

$\mathcal{TC}$. The set of topology change events. Formally,

$= \{LinkFailure_u(v), LinkRecovery_u(v,c), LinkCostChange_u(v,c) : (u,v) \in LINKS \wedge c \in I^+\}$.

Conventions: We use the term **distance** when we refer to the values of state variables $Dist_u(z)$ and $Distvia_u(v,z)$, either in the nodes or in transit in the channels. We say "distance d in transit for destination z" to mean there is a message in transit whose d_vector contains a (z,d) pair. We use the term **cost**, and not "distance", when we refer to the current values of link costs, e.g. $Path\_cost$, $Cost$. Note that costs can not change unless a topology change happens.

Notation: For any non–empty sequence $\langle x_0, \ldots, x_n \rangle$, $last(\langle x_0, \ldots, x_n \rangle)$ denotes x_n, $tail(\langle x_0, \ldots, x_n \rangle)$ denotes $\langle x_1, \ldots, x_n \rangle$, and $head(\langle x_0, \ldots, x_n \rangle)$ denotes x_0. When applied to a null sequence, $head(\langle\rangle) = last(\langle\rangle) = nil$ and $tail(\langle\rangle) = \langle\rangle$. We use @ as the concatenation operator for sequences, i.e. $\langle x_0, \ldots, x_n \rangle @ \langle y_0, \ldots, y_m \rangle = \langle x_1, \ldots, x_n, y_0, \ldots, y_m \rangle$.

We define a boolean function $Has\_optimal\_path(u,z)$ that is true iff the next–hop path starting from u reaches z and has optimal cost; note that this implies that all nodes on the next–hop path also have optimal next–hop paths to z. Formally:

$$\begin{aligned} Has\_optimal\_path(u,z) \equiv\ & last(Nhoppath(u,z)) = z \\ & \wedge [\forall x \in Nhoppath(u,z) : Dist_x(z) = Cost(x,z) = Path\_cost(Nhoppath(x,z))] \end{aligned}$$

The desired objective can be stated as follows, where A is some state formula (that can depend on the routing algorithm):

- $Symmetric \wedge (u,z) \in Reachable$ *leads–to* $\mathcal{TC} \vee (u,z) \in Reachable \wedge Has\_optimal\_path(u,z) \wedge A$
- $Symmetric \wedge (u,z) \in Reachable \wedge Has\_optimal\_path(u,z) \wedge A$ *unless* $\mathcal{TC}$

That is, after any succession of topology changes that leaves the network in a symmetric state, if there are no further topology changes, then every reachable node u eventually achieves a stable optimal path to z. We point out that most routing algorithms, including the ones in this paper, do not satisfy the above property if $A = true$. That is, it is possible for a node to achieve an optimal next–hop path and then switch to some other non–optimal path. However, eventually, it will find an optimal next–hop path and also satisfy A; once this is achieved, the optimal next–hop path is stable.

The following assertions M_1 and M_2 specify an appropriate A for algorithm **A1**:

(M_1) $Symmetric$ *leads–to* $\mathcal{TC} \vee$

$$[\forall (u,z) \in Reachable : Has\_optimal\_path(u,z) \wedge [\forall v \in neighbors(u) : Channel_{uv}(z) = \langle\rangle]]$$

(M_2) $Symmetric \wedge [\forall (u,z) \in Reachable : Has\_optimal\_path(u,z) \wedge [\forall v \in neighbors(u) : Channel_{uv}(z) = \langle\rangle]]$

$\quad unless\ \mathcal{TC}$

where $Channel_{uv}(z)$ is a state function which denotes the sequence of messages in $Channel_{uv}$ that contain a distance for destination z. Formally,

$$\begin{aligned}
&Channel_{uv}(z) = \langle m_0, m_1, \ldots, m_n \rangle \text{ such that} \\
&\quad [\exists p_0, \ldots, p_{n+1} : Channel_{uv} = p_0 @ \langle m_0 \rangle @ p_1 @ \langle m_1 \rangle @ \ldots @ p_n @ \langle m_n \rangle @ p_{n+1} \ \wedge \\
&\qquad [\forall i, 0 \leq i \leq n, \exists d : (z,d) \in m_i] \ \wedge \\
&\qquad [\forall i, 0 \leq i \leq n+1, \forall m \in p_i, \forall d : (z,d) \notin m]].
\end{aligned}$$

Theorem 1. **A1** *satisfies* M_1 *and* M_2.

Proof of Theorem 1

Readers who are interested in the algorithms but not in the proofs can skip this proof.

Conventions: For a leads-to assertion "A *leads–to* $\mathcal{TC} \vee B$", we refer to A as the left side of the assertion, and B as the right side. We use the same convention for "A *unless* $\mathcal{TC} \vee B$" and for "*Invariant* $A \Rightarrow B$". Most of our leads-to assertions have the form $Symmetric \wedge A$ *leads–to* $\mathcal{TC} \vee B$, that is, if $Symmetric$ and A holds, then eventually B holds or a topology change occurs. When informally describing such an assertion, for brevity, we just say "if A holds then eventually B holds". The same convention is used with assertions of the form "$Symmetric \wedge A$ *unless* $\mathcal{TC} \vee B$". We assume the following precedence of operators: $\neg$, $\wedge$, $\vee$, $\Rightarrow$, *Invariant*, *unless*, *leads–to*. We say cost of a node pair (u,z) and distance of a node pair (u,z) to mean $Cost(u,z)$ and $Dist_u(z)$ respectively.

The following assertions express rather obvious relationship between neighboring nodes:

(B_1) $Dist_v(z) = d \wedge (v,u) \in UPLINKS$ *leads–to* $\mathcal{TC} \vee Distvia_u(v,z) = d + Linkcost_u(v)$

(B_2) *Invariant* $(v,u) \in UPLINKS \wedge Channel_{vu}(z) \neq \langle\rangle \Rightarrow (z, Dist_v(z)) = last(Channel_{vu}(z))$

(B_3) *Invariant* $(v,u) \in UPLINKS \wedge Channel_{vu}(z) = \langle\rangle \Rightarrow Distvia_u(v,z) = Dist_v(z) + Linkcost_u(v)$

(B_4) *Invariant* $(v,u) \in UPLINKS \wedge Distvia_u(v,z) \neq Dist_v(z) + Linkcost_u(v)$

$\quad \Rightarrow (z, Dist_v(z)) = last(Channel_{vu}(z))$

(B_5) m in $Channel_{vu}$ *leads–to* $\mathcal{TC} \vee m = front(Channel_{vu})$

(B_6) $Channel_{uv} = \langle m \rangle @ x$ *leads–to* $\mathcal{TC} \vee [\exists y : Channel_{uv} = x @ y]$

(B_7) (z,d) in $Channel_{vu}$ *leads–to* $\mathcal{TC} \vee Distvia_u(v,z) = d + Linkcost_u(v)$

(B_8) $(z,d) = front(Channel_{vu})$ *leads–to* $\mathcal{TC} \vee Distvia_u(v,z) = d + Linkcost_u(v)$

B_1, B_2, B_3, B_4, B_7 and B_8 deal with the distances of neighboring nodes to a destination z and the distances to z in transit between the neighboring nodes.

B_1 states that if the distance of v is d and the link (v, u) is not failed, then u eventually learns of d. B_1 follows from B_4 and B_7 by the closure.

B_2 states that if a channel has distances to z, then the last message contains the current distance of the sender. B_2 follows from invariance rule.

B_3 states that if no distances to z are in transit, then the distance of the receiver through the sender is up-to-date. B_3 follows from B_2 using invariance rule.

B_4 states that if a distance of node u via a neighbor v is not up-to-date, then the current distance of v is in the last message in $Channel_{vu}(z)$. B_4 follows from B_2 and B_3 by implication (left side of B_4 implies the negation of the right side of B_3; since B_3 holds, the left side of B_3 must also be false, which implies the left side of B_2, which implies the right side of B_2, which implies the right side of B_4).

B_5 states that a message in transit eventually advances to the front of the channel. B_6 states that the message in the front of the channel eventually gets removed. B_6 follows from leads–to rule (via receive event). B_5 follows from B_6 by closure. B_7 states that each distance in link (v, u) is eventually used to update the distance of u via v. B_8 states that the distance in the front of a link (v, u) is eventually used to update the distance of u via v. B_8 follows from leads–to rule (via receive event). B_7 follows from B_5 and B_8 by the closure.

The following safety assertions state that the values of *Symmetric*, *Reachable*, cost of a node pair, and *HighestCost* do not change. Each of them holds from the unless rule.

(C_1) $Symmetric$ *unless* $\mathcal{TC}$

(C_2) $Reachable = \mathcal{S}$ *unless* $\mathcal{TC}$

(C_3) $Cost(u, z) = K$ *unless* $\mathcal{TC}$

(C_4) $HighestCost = K$ *unless* $\mathcal{TC}$

We now define functions that, in some sense, characterize the essence of algorithm **A1**:

In. Maximal subset of *Reachable* such that (u, z) is a member of *In* iff

(1) $Has\_optimal\_path(u, z)$,

(2) for any message (x, d) in transit, $Dist_u(z)$ is less than d,

(3) for any node pair (w, x) in *Reachable* not satisfying $Has\_optimal\_path(w, x)$, $Dist_u(z) < Dist_w(x)$ and $Dist_u(z) < Cost(w, x)$.

$Out = Reachable - In$.

Lowest. The minimum of the cost of node pairs in Out, the distances of node pairs in Out, and the distances in transit between nodes from which the destination is reachable. Formally,

$$= \min(\{Cost(x,z) : (x,z) \in Out\} \cup$$
$$\{Dist_u(z) : (u,z) \in Out\} \cup$$
$$\{d : (x,d) \in Channel_{uv} \ \wedge \ (u,x),(v,x) \in Reachable\}).$$

The intuition behind a node pair (u,z) being in In is the following: u has an optimal path to z, and this cannot be affected by any message in transit or by any message that can be generated by other nodes. Note that if a node pair (u,z) is in In and $u \neq z$, then $NHop_u(z) \neq nil$ and the node pair $(NHop_u(z), z)$ is also in In. If a node pair (u,z) is in In, then the outgoing channels of u do not contain any (z,d) messages. This follows from B_2 and the definition of In (i.e. since (u,z) is in In, the messages in transit for z have larger distances than the distance of u, and if an outgoing channel of u contained a message for z, the last message in that channel for z would contain a distance which was not larger).

The intuition behind $Lowest$ is the following: $Lowest$ never decreases, and keeps increasing as long as it is less than $HighestCost$. Furthermore, $Lowest > HighestCost$ iff $In = Reachable$ (this is because $Lowest > HighestCost$ means that cost of all reachable node pairs are less than $Lowest$, hence they are not in Out). In contrast, the minimum distance in transit can decrease or increase without a change in Out; the same is true for the minimum distance of a node pair in Out .

We now proceed to prove M_1 and M_2. The proof of M_1 is summarized in Figure 1.

M_2 holds from the unless rule; specifically, once the left side of M_2 holds, no receive event of any node in $Reachable$ is enabled, and all other events belong to $\mathcal{TC}$. Thus, it suffices to prove M_1.

(M_3) $Symmetric$ *leads–to* $\mathcal{TC} \vee In = Reachable$

M_3 states that eventually In contains all reachable node pairs. M_1 follows from M_3 by closure (since $In = Reachable$ implies right side of M_1). Thus it suffices to prove M_3.

(M_4) $Symmetric$ *leads–to* $\mathcal{TC} \vee Lowest > HighestCost$

M_4 states that $Lowest$ eventually exceeds $HighestCost$. M_3 follows from M_4 by closure. Thus it suffices to prove M_4.

(M_5) $Symmetric \wedge Lowest = k \leq HighestCost$ *leads–to* $\mathcal{TC} \vee Lowest \geq k+1$

M_4 follows from M_5, C_1 and C_4 by closure. Thus it suffices to prove M_5. We first define the following functions:

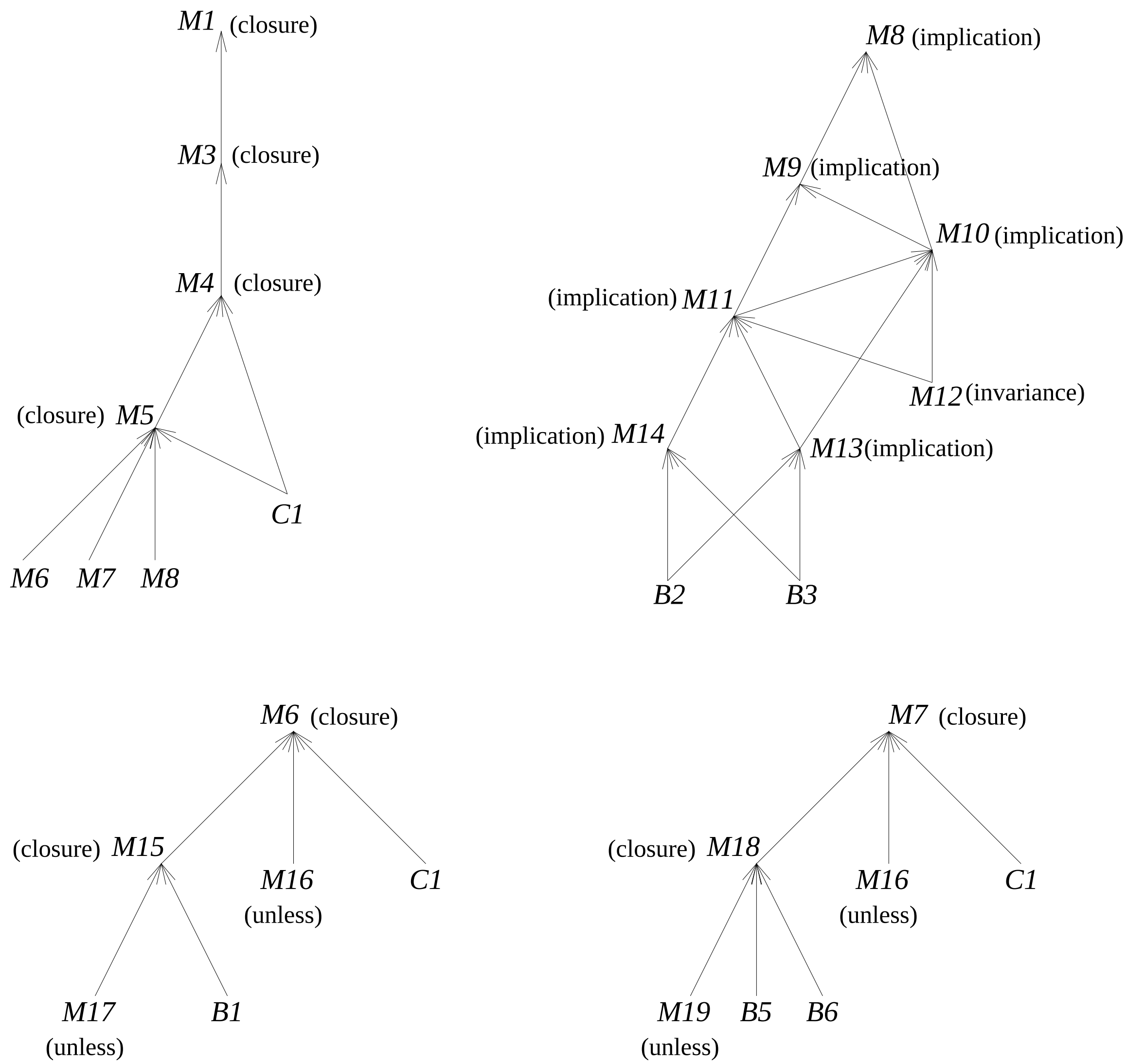

Figure 1: Proof of $M1$. Each arrow indicates that the tail assertion is used in the proof of the head assertion. Proof rule used is indicated in parenthesis.

$$DVia(k) = \{(u, v, z) : Distvia_u(v, z) = k \ \wedge \ (v, z) \in Out\}$$

$$DTransit(k) = \text{bag}\{(u, v, z) : (z, k) \in Channel_{uv} \ \wedge \ (u, z), (v, z) \in Reachable\}$$

Note that $DTransit(k)$ is a bag; i.e. if there are two messages whose distance vectors contain the same (z, k) pair in the same channel, $DTransit(k)$ contains two $\langle u, v, z\rangle$ triplets.

We next define the following assertions:

(M_6) $\mathit{Symmetric} \wedge \mathit{Lowest} = k$ leads–to $\mathcal{TC} \vee \mathit{Lowest} \geq k \wedge |\mathit{DVia}(k)| = 0$

(M_7) $\mathit{Symmetric} \wedge \mathit{Lowest} \geq k \wedge |\mathit{DVia}(k)| = 0$

leads–to $\mathcal{TC} \vee \mathit{Lowest} \geq k \wedge |\mathit{DVia}(k)| = 0 \wedge |\mathit{DTransit}(k)| = 0$

(M_8) Invariant $\mathit{Lowest} \geq k \wedge |\mathit{DVia}(k)| = 0 \wedge |\mathit{DTransit}(k)| = 0 \Rightarrow \mathit{Lowest} > k$

M_6 and M_7 state that if $\mathit{Lowest} = k$, then $\mathit{DVia}(k)$ and $\mathit{DTransit}(k)$ eventually become empty. At that point, M_8 states that Lowest is greater than k. M_5 follows from M_6, M_7, M_8 and C_1 by closure.

Thus it suffices to prove M_6, M_7 and M_8, which is done next.

Proof of M_8

The following assertions state that if $\mathit{Lowest} \geq k$ and $\mathit{DVia}(k)$ and $\mathit{DTransit}(k)$ are empty, node pairs in Out have both costs and distances higher than k.

(M_9) Invariant $\mathit{Lowest} \geq k \wedge |\mathit{DVia}(k)| = 0 \wedge |\mathit{DTransit}(k)| = 0 \wedge \mathit{Cost}(u,z) = k \Rightarrow (u,z) \in \mathit{In}$

(M_{10}) Invariant $\mathit{Lowest} \geq k \wedge |\mathit{DVia}(k)| = 0 \wedge (u,z) \in \mathit{Reachable} \wedge \neg \mathit{Has\_optimal\_path}(u,z) \Rightarrow \mathit{Dist}_u(z) > k$

M_8 follows from M_9 and M_{10} by implication. (The details are as follows: From M_9, the cost of a node pair in Out is greater than k. From $|\mathit{DTransit}(k)| = 0$ and $\mathit{Lowest} \geq k$, the minimum distance in transit is greater than k. From M_{10}, the distance of a node pair in Out is greater than k; note that if (u,z) is in Out and $\mathit{Has\_optimal\_path}(u,z)$, then $\mathit{Dist}_u(z) = \mathit{Cost}(u,z) > k$. Hence Lowest is greater than k.)

Thus it suffices to prove M_9 and M_{10}. We next proceed to prove M_9.

(M_{11}) Invariant $\mathit{Lowest} \geq k \wedge |\mathit{DVia}(k)| = 0 \wedge \mathit{Cost}(u,z) = k \Rightarrow \mathit{Has\_optimal\_path}(u,z)$

M_9 follows from M_{11} and M_{10} by implication. (The details are as follows: Consider a node pair (u,z) satisfying left side of M_9. (u,z) has optimal path (from M_{11}). Distances of node pairs (w,x) not satisfying $\mathit{Has\_optimal\_path}(w,x)$ are greater than k (from M_{10}). Costs of node pairs (w,x) not satisfying $\mathit{Has\_optimal\_path}(w,x)$ are greater than k (from M_{11} and $\mathit{Lowest} \geq k$). Distances in transit are greater than k (from $\mathit{Lowest} \geq k$ and $|\mathit{DTransit}(k)| = 0$).)

Thus it suffices to prove M_{11} and M_{10}. We next proceed to prove M_{11}.

(M_{12}) Invariant $u \neq z \Rightarrow \mathit{Dist}_u(z) = \min\{\mathit{Distvia}_u(v,z) : v \in \mathit{neighbors}(u)\}$

(M_{13}) Invariant $[\forall (v,z) \in \mathit{In} \wedge v \in \mathit{neighbors}(u) : \mathit{Distvia}_u(v,z) \geq \mathit{Cost}(u,z)]$

(M_{14}) Invariant $\mathit{Cost}(u,z) \leq \mathit{Lowest} \wedge u \neq z \Rightarrow [\exists (v,z) \in \mathit{In} \wedge v \in \mathit{neighbors}(u) : \mathit{Distvia}_u(v,z) = \mathit{Cost}(u,z)]$

M_{12} states that distance of a node pair equals the minimum of distances via its neighbors. M_{12} follows from invariance rule.

M_{13} states that distance of a node via a node pair in In is greater than or equal to the cost of the node. M_{13} follows from B_2 and B_3 by implication (since (v,z) is in In, $Dist_v(z) = Cost(v,z)$ and v's outgoing channels do not contain a message for z, hence $Distvia_u(v,z)$ equals $Cost(v,z) + Linkcost_u(v)$, which is greater than or equal to cost of node pair (u,z)).

M_{14} states that a node pair with cost less than or equal to $Lowest$ has a neighbor in In and its distance via this neighbor equals its cost. M_{14} follows from B_2 and B_3 by implication (note that if cost of (u,z) is less than or equal to $Lowest$ and v is u's next node on an optimal path, then (v,z) is in In since v has a smaller cost; also since outgoing channels of v do not contain a message for z, the distance of u via v equals cost of (u,z)).

M_{11} follows from M_{14}, M_{13} and M_{12} by implication. (The details are as follows: Consider node pair (u,z) that satisfies left side of M_{11}. From M_{14}, there is a neighbor v of u such that (v,z) is in In and u's distance to z via v equals its cost k. From M_{13}, u's distance to z via neighbors in In is not less than k. From $|DVia(k)| = 0$ and $Lowest \geq k$ (left side of M_{11}), u's distance to z via neighbors in Out is higher than k. Thus from M_{12}, u's distance to z is k and $Nhop_u(z)$ is a neighbor v in In. Thus u has optimal path.)

Thus it suffices to prove M_{10}. M_{10} follows from M_{11}, M_{12} and M_{13} by implication. (The details are as follows: Consider a node pair (u,z) that satisfies left side of M_{10}. From $\neg Has\_optimal\_path(u,z)$ and $Lowest \geq k$, we have $Cost(u,z) \geq k$. From $\neg Has\_optimal\_path(u,z)$ and M_{11}, we have $Cost(u,z) \neq k$. Thus, $Cost(u,z) > k$. From M_{13}, u's distance to z via neighbors in In is greater than k. From $|DVia(k)| = 0$ and $Lowest \geq k$ (left side of M_{10}), u's distance to z via neighbors in Out is greater than k. Thus, from M_{12}, u's distance to z is greater than k.)

This completes the proof of M_8.

Proof of M_6

We repeat M_6:

(M_6) $Symmetric \wedge Lowest = k$ *leads–to* $\mathcal{TC} \vee Lowest \geq k \wedge |DVia(k)| = 0$

Define

(M_{15}) $Symmetric \wedge Lowest \geq k \wedge |DVia(k)| = n > 0$ *leads–to* $\mathcal{TC} \vee |DVia(k)| = n - 1$

(M_{16}) $Lowest \geq k$ *unless* $\mathcal{TC}$

M_{16} follows from the unless rule. M_6 follows from M_{15}, M_{16} and C_1 by closure. Thus it suffices to prove M_{15}.

(M_{17}) $Lowest \geq k \wedge |DVia(k)| \leq n$ *unless* $\mathcal{TC}$

M_{17} states that if $Lowest \geq k$ then the size of $DVia(k)$ does not increase. M_{17} follows from the unless rule.

M_{15} follows from M_{17} and B_1 by closure. (The details are as follows: From B_1, $Distvia_u(v,z)$ eventually becomes greater than $Lowest$ (since d in B_1 is greater than $Lowest$), hence decreases $|DVia(k)|$. M_{17} ensures that $|DVia(k)|$ does not increase before $Distvia_u(v,z)$ becomes greater than $Lowest$.)

This completes the proof of M_6.

Proof of M_7

We repeat M_7:

$$(M_7)\ \ Symmetric \ \wedge\ Lowest \geq k \ \wedge\ |DVia(k)| = 0$$
$$\textit{leads–to}\ \ \mathcal{TC} \ \vee\ Lowest \geq k \ \wedge\ |DVia(k)| = 0 \ \wedge\ |DTransit(k)| = 0$$

Define

$$(M_{18})\ \ Symmetric \ \wedge\ Lowest \geq k \ \wedge\ |DVia(k)| = 0 \ \wedge\ |DTransit(k)| = n > 0$$
$$\textit{leads–to}\ \ \mathcal{TC} \ \vee\ |DVia(k)| = 0 \ \wedge\ |DTransit(k)| = n-1$$

M_{18} states that if $Lowest \geq k$ and $DVia(k)$ is empty, then the size of $DTransit(k)$ eventually decreases. M_7 follows from M_{18}, M_{16} and C_1 by closure. Thus it suffices to prove M_{18}.

$$(M_{19})\ \ Lowest \geq k \ \wedge\ |DVia(k)| = 0 \ \wedge\ DTransit(k) \text{ bag-subset S } \ \textit{unless}\ \ \mathcal{TC}$$

M_{19} states that if $Lowest \geq k$ and $DVia(k)$ is empty then $DTransit(k)$ does not expand[1]. M_{19} follows from unless rule.

M_{18} follows from B_5, B_6 and M_{19} by closure. (The details are as follows: From B_5, a message participating in $DTransit(k)$ advances to front. From B_6, it gets removed, decreasing $|DTransit(k)|$. M_{19} ensures that $DTransit(k)$ does not expand while the message advances to front.)

This completes the proof of M_7, and hence of Theorem 1.

End of proof of Theorem 1

Even though we have shown that after any succession of topology changes, the nodes that can reach the destination obtain optimal paths, this convergence may contain long–lived loops and be very lengthy. For example, consider the simple network in Figure 2.a. Three are three nodes u, v, and z. Destination node is z. Assume all link costs are 1. Numbers on the arrows indicate the distances of nodes via their neighbors, and solid arrows indicate the next-hops to z. That is, node u's distance to z via z is 1 and via node v is 3. In Figure 2.b, cost of the link (u,z) increases to D such that $D > 3$. As a result u chooses v as its

[1] Bag S is a bag-subset of bag Z iff every element m of S is also an element of Z. Note that, if S contains k instances of m, then Z contains at least k instances of m.

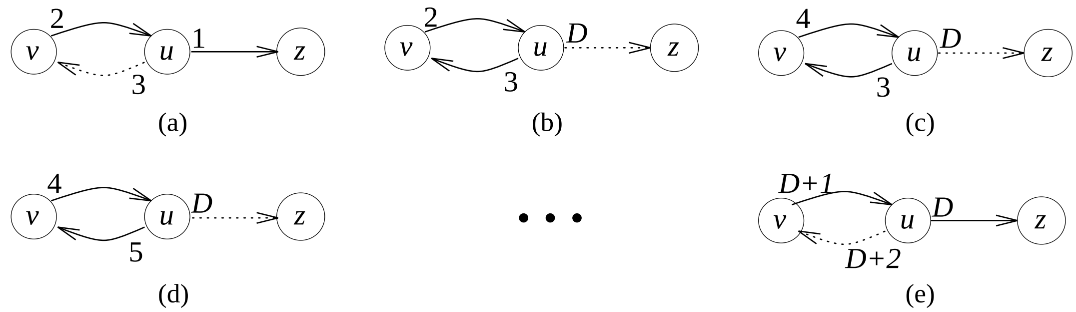

Figure 2: Long-lived loops.

next-hop, causing a loop, and sends its new distance to v. Upon receiving this message, v will add 1 (i.e. $Linkcost_v(u)$) to this distance (see Figure 2.c), and send a message back to u, causing node u's distance to increase (see Figure 2.d). Node u and v will keep on exchanging messages (referred to as *bouncing effect* in the literature), each time increasing their distances by 2 (i.e. $Linkcost_u(v) + LinkCost_v(u)$) until node u's distance via v exceeds D, at which point u chooses z as its next-hop. This convergence will require $(D-1)/(Linkcost_u(v) + LinkCost_v(u))$ number of exchanges. Note that if D is ∞ (i.e. link (u, z) fails, and thus nodes u and v cannot reach z), they will exchange distances indefinitely (referred to as *count–to–infinity problem*). With more realistic network topologies, this behavior can be even more complex, for example: loops can involve multiple hops and breaking one loop may cause another loop. In the next algorithm, these problems are avoided.

4 Algorithm A2

Table 2 specifies the state variables and events of a node in **A2**. (Refer to the table in the following discussion.) Each node maintains the state variables required for **A1**. In addition, node u maintains in state variable $Routevia_u(v, z)$ an estimate of the next–hop path for destination z via neighbor v. It is equal to the null sequence if node u believes z cannot be reached via v. Node u also maintains a state variable $Costseqvia_u(v, z)$ which stores the sequence of estimated link costs for the corresponding links in $Routevia_u(v, z)$. State variables $Route_u(z)$ and $Costseq_u(z)$ store the route and cost sequence via node u's next–hop.

Convention: We use the term **route** to refer to estimates maintained by nodes of next–hop paths.

The variables $Costseq_u(z)$ and $Costseqvia_u(v, z)$ are *auxiliary* variables; they are needed for verification only, and do not have to be implemented. (Formally they satisfy the following conditions: (1) they do not affect the enabling condition of any event, and (2) they do not affect the update of any nonauxiliary state variable [16].)

Algorithm **A2** is like algorithm **A1**, except that **A2** uses paths to avoid long–lived loops. Long–lived loops in the next–hop path for destination z can be avoided by having node u not choose a neighbor v as its next–hop if $Routevia_u(v,z)$ contains a cycle. Another way to achieve the same effect is by having node v send ∞ as its distance to node u if node u is in $Route_v(z)$. We have chosen the second approach, as specified in the last five lines of procedure *Update&Send* in table 2. That is, sending ∞ as the distance prevents the receiver from choosing a route with a loop. It does not prevent the receiver from choosing an optimal path.

In addition to exchanging distances, nodes also exchange information about their paths and cost sequences. More precisely, node v sends messages of the form (v, d_vector), where d_vector is a set of (z, d, p, cs, rd) tuples such that either (1) $d = rd = Dist_v(z)$, $p = Route_v(z)$, and $cs = Costseq_v(z)$ if node u is not in $Route_v(z)$, or (2) $d = \infty$, $p = \langle\rangle$, $cs = \langle\rangle$, and $rd = Dist_v(z)$, if node u is in $Route_v(z)$. Fields cs and rd are auxiliary fields, and do not have to be implemented (rd is only used in the proof of Theorem 2).

When $Linkcost_u(v)$ changes, $Distvia_u(v,z)$, $Routevia_u(v,z)$, $Costseqvia_u(v,z)$, and if needed $Nhop_u(z)$, $Dist_u(z)$, $Route_u(v,z)$, and $Costseq_u(z)$, are updated for each destination z (as shown in procedure *Update&Send* in table 2). If the distance or route of any destination has been affected, node u sends messages to its neighbors for all affected destinations z (as described in the previous paragraph).

When node u receives a (v, d_vector) message, it updates its state variables for each destination z in d_vector (note that rd is not used to update any state variable). If the distance or route of any destination has been affected, node u sends messages to its neighbors.

Theorem 2. **A2** *satisfies* M_1 *and* M_2.

Proof of Theorem 2

The proof of Theorem 2 is identical to that of Theorem 1, except that the assertions B_1-B_8 describing the relationship between neighboring nodes, are replaced by new assertions B_1-B_9 below.

The main differences between **A2** and **A1** are reflected in the new B assertions. First, messages in transit may contain ∞ as distance even though the sender's distance is finite (see B_2 below). This only happens when the receiver is on the sender's route. Second, when the channel between two nodes do not contain a distance for a destination, distance of the receiver via the sender may not equal the sum of sender's distance and the cost of the link between them (see B_3 below). B_4 now has two parts B_{4a} and B_{4b}; the first part covers the case when the receiver is not on the route of the sender, and the second part covers the case when it is. Proofs of B_1–B_8 are identical to their counterparts in **A1**. (Assertions B_5 and B_6 stay the same.)

(B_1) $Dist_v(z) = d \;\wedge\; Route_v(z) = p \;\wedge\; (v,u) \in UPLINKS$

$$\textit{leads–to}\;\; \mathcal{TC} \vee (u \notin p \;\Rightarrow\; Distvia_u(v,z) = d + Linkcost_u(v)) \;\vee$$
$$(u \in p \;\Rightarrow\; Distvia_u(v,z) = \infty)$$

(B_2) Invariant $Channel_{vu}(z) \neq \langle\rangle \;\wedge\; (u,v) \in UPLINKS$

$$\Rightarrow ((z, Dist_v(z), Route_v(z), Costseq_u(z), Dist_v(z)) = last(Channel_{vu}(z)) \;\wedge\; u \notin Route_v(z)) \;\vee$$
$$((z, \infty, \langle\rangle, \langle\rangle, Dist_v(z)) = last(Channel_{vu}(z)) \;\wedge\; u \in Route_v(z))$$

(B_3) Invariant $Channel_{vu}(z) = \langle\rangle \;\wedge\; (u,v) \in UPLINKS$

$$\Rightarrow (Distvia_u(v,z) = Dist_v(z) + Linkcost_u(v) \;\wedge\; u \notin Route_v(z)) \;\vee$$
$$(Distvia_u(v,z) = \infty \;\wedge\; u \in Route_v(z))$$

(B_{4a}) Invariant $(u,v) \in UPLINKS \wedge Distvia_u(v,z) \neq Dist_v(z) + Linkcost_u(v) \;\wedge\; u \notin Route_v(z)$

$$\Rightarrow\; (z, Dist_v(z), Route_v(z), Costseq_u(z), Dist_v(z)) = last(Channel_{vu}(z))$$

(B_{4b}) Invariant $(u,v) \in UPLINKS \wedge Distvia_u(v,z) \neq \infty \;\wedge\; u \in Route_v(z)$

$$\Rightarrow\; (z, \infty, \langle\rangle, \langle\rangle, Dist_v(z)) = last(Channel_{vu}(z))$$

(B_7) (z,d,p,cs,rd) in $Channel_{vu}$ *leads–to* $\mathcal{TC} \;\vee\; Distvia_u(v,z) = d + Linkcost_u(v)$

(B_8) $(z,d,p,cs,rd) = front(Channel_{vu})$ *leads–to* $\mathcal{TC} \;\vee\; Distvia_u(v,z) = d + Linkcost_u(v)$

(B_9) Invariant (z,d,p,cs,rd) in $Channel_{vu} \;\Rightarrow\; d = \infty \;\vee\; d = rd$

B_9 states that rd in a message is less than or equal to the corresponding d. B_9 follows from invariance rule.

We redefine *In*, *Lowest* and *DTransit* for **A2** as follows:

In. Maximal subset of *Reachable* such that (u,z) is a member of *In* iff

(1) $Has\_optimal\_path(u,z)$,

(2) for any message (x,d,p,cs,rd) in transit, $Dist_u(z)$ is less than rd,

(3) for any node pair (w,x) in *Reachable* not satisfying $Has\_optimal\_path(w,x)$,

$Dist_u(z) < Dist_w(x)$ and $Dist_u(z) < Cost(w,x)$.

Lowest. Formally,

$$= \min(\{Cost(x,z) : (x,z) \in Out\} \cup$$
$$\{Dist_u(z) : (u,z) \in Out\} \cup$$
$$\{rd : (x,d,p,cs,rd) \in Channel_{uv} \;\wedge\; (u,x),(v,x) \in Reachable\}).$$

$$DTransit(k) = \text{bag}\{(u,v,z) : (z,d,p,cs,rd) \in Channel_{uv} \;\wedge\; rd = k \;\wedge\; (u,z),(v,z) \in Reachable\}.$$

The proof of Theorem 2 is identical to the proof of Theorem 1 with new B assertions. B_9 is required for M_{17}, M_{18} and M_{19} to hold for **A2**. Except for these changes, every assertion used in the proof of Theorem 1 also holds for **A2** (and the proof is identical). Hence M_1 and M_2 hold for **A2**.

End of proof of Theorem 2

Next, we establish that after any succession of topology changes that leaves the network symmetric, **A2** achieves optimal paths within $N + H$ steps assuming synchronous execution.

We define a **synchronous execution** as follows: Each message includes a *step counter* which is a non–negative integer. Any message sent by a receive event has step counter one higher than the step counter of the received message. Any topology change event sets the step counter of all messages (including the ones being generated) to zero. We require that *Receive* events are executed such that the sequence of step counters of the received messages is non–decreasing. Formally, we define *Step* to be the step counter of the last message received, and add the following *SE* condition as a conjunct to the enabling condition of every receive event:

$$SE: \text{ step counter of the message to be received} = \text{ minimum step counter of the messages in transit}$$

Note that *Step* equals 0 immediately after any topology change.

The following assertions N_1 and N_2 state the desired property, that is, reachable node pairs achieve optimal paths within $N + H$ steps, and other node pairs obtain ∞ distances within N steps.

$$(N_1)\ \ Symmetric \ \wedge\ Step = 0 \ \ \textit{leads–to}\ \ \mathcal{TC} \ \vee\ Step \leq N + H \ \wedge$$
$$[\forall (u,z) \in Reachable : Has\_optimal\_path(u,z) \ \wedge\ [\forall v \in neighbors(u) : Channel_{uv}(z) = \langle\rangle]]$$

$$(N_2)\ \ Symmetric \ \wedge\ Step = 0 \ \ \textit{leads–to}\ \ \mathcal{TC} \ \vee\ Step \leq N \ \wedge\ [\forall (u,z) \notin Reachable : Dist_u(z) = \infty]$$

Theorem 3. *Assuming synchronous execution,* **A2** *satisfies* N_1 *and* N_2*.*

Proof of Theorem 3

The rest of Section 4 is a proof of Theorem 3. Readers interested in the algorithms but not in the proofs can skip to Section 5.

Conventions: We use *step#* to refer to the step number of a message.

We recast the assertions relating the states of neighbor nodes assuming synchronous execution:

$$(D_1)\ \ Step = n \ \wedge\ (v,u) \in UPLINKS \ \wedge\ Dist_v(z) = d \ \wedge\ Route_v(z) = p \ \wedge\ Costseq_v(z) = cs \ \wedge\ u \notin p$$
$$\textit{leads–to}\ \ \mathcal{TC} \vee Step \leq n+1 \ \wedge\ Distvia_u(v,z) = d + Linkcost_u(v)$$
$$\wedge\ Routevia_u(v,z) = \langle u\rangle @ p \ \wedge\ Costseqvia_u(v,z) = \langle Linkcost_u(v)\rangle @ cs$$

(D_2) $Step = n \ \wedge \ (u, v) \in UPLINKS \ \wedge \ Dist_v(z) = d \ \wedge \ Route_v(z) = p \ \wedge \ Costseq_v(z) = cs \ \wedge \ u \in p$

$$\textit{leads–to} \ \ \mathcal{TC} \vee Step \leq n+1 \ \wedge \ Distvia_u(v, z) = \infty$$
$$\wedge \ Routevia_u(v, z) = \langle\rangle \ \wedge \ Costseqvia_u(v, z) = \langle\rangle$$

(D_3) **Invariant** $Step = n \ \wedge \ (u, v) \in UPLINKS \ \wedge$

$$u \notin Route_v(z) \ \wedge \ Distvia_u(v, z) \neq Dist_v(z) + Linkcost_u(v) \ \wedge \ Routevia_u(v, z) \neq \langle u\rangle @ Route_v(z) \ \Rightarrow$$
$$\textit{last}(Channel_{vu}(z)) = (z, Dist_v(z), Route_v(z), Costseq_v(z), Dist_v(z)) \text{ with } n \leq \textit{step\#} \leq n+1$$

(D_4) **Invariant** $Step = n \ \wedge \ (u, v) \in UPLINKS \ \wedge$

$$u \in Route_v(z) \ \wedge \ Distvia_u(v, z) \neq \infty \ \wedge \ Routevia_u(v, z) \neq \langle\rangle \ \Rightarrow$$
$$\textit{last}(Channel_{vu}(z)) = (z, \infty, \langle\rangle, \langle\rangle) \text{ with } n \leq \textit{step\#} \leq n+1$$

(D_5) $\textit{front}(Channel_{vu}) = (z, d, p, cs, rd)$ with $\textit{step\#} = n \ \wedge \ d \neq \infty$

$$\textit{leads–to} \ \ \mathcal{TC} \vee Step = n \ \wedge \ Distvia_u(z) = d + Linkcost_u(v)$$
$$\wedge \ Routevia_u(v, z) = \langle u\rangle @ p \ \wedge \ Costseqvia_u(v, z) = \langle Linkcost_u(v)\rangle @ cs$$

(D_6) $\textit{front}(Channel_{vu}) = (z, \infty, p, cs, rd)$ with $\textit{step\#} = n$

$$\textit{leads–to} \ \ \mathcal{TC} \vee Step = n \ \wedge \ Distvia_u(z) = \infty$$
$$\wedge \ Routevia_u(v, z) = \langle\rangle \ \wedge \ Costseqvia_u(v, z) = \langle\rangle$$

Suppose link (v, u) is not failed. Given any state of v's distance, route and cost sequence to z, D_1 states that if u is not on the route from v to z, then u eventually learns of v's state within one step. D_2 states that if u is on the route from v to z, then u eventually learns within one step that v has a distance of ∞, route of $\langle\rangle$, and cost sequence of $\langle\rangle$. D_3 and D_4 (and D_5 and D_6) make the same distinction. D_3 and D_4 follow from invariance rule. D_5 and D_6 follows from the leads–to rule (via receive event). D_1 follows from D_3, B_5, and D_5 by closure. D_2 follows from D_4, B_5, and D_6 by closure.

Define $\langle x_0, \ldots, x_n\rangle$ to be a *ud-path* from x_0 to x_n if $[\forall 0 \leq i < n : (x_i, x_{i+1}) \in LINKS]$. Note that *ud-path* does not distinguish between up and down links.

Some safety assertions:

(E_1) **Invariant** $Route_u(z)$ is a simple *ud-path*

$Routevia_u(v, z)$ is a simple *ud-path*

(z, d, p, cs, rd) in $Channel_{vu} \ \Rightarrow \ p$ is a simple *ud-path* $\wedge \ u \notin p$

(E_2) **Invariant** $|Routevia_u(v, z)| \leq N \ \wedge \ |Route_u(z)| \leq N \ \wedge \ [(z, d, p, cs, rd) \text{ in transit} \ \Rightarrow \ |p| \leq N]$

(E_3) *Invariant* $Distvia_u(v,z) = \text{sum } \{c : c \in Costseqvia_u(v,z)\}$

$\wedge\ Dist_u(z) = \text{sum } \{c : c \in Costseq_u(z)\}$

$\wedge\ [(z,d,p,cs,rd) \text{ in transit } \Rightarrow\ d = \text{sum } \{c : c \in cs\}]$

E_1 follows from invariance rule. E_2 states that route lengths (in number of links) are bounded above by N. E_2 follows from E_1 by implication (since a simple path may contain at most N nodes).

E_3 states that all distances equal the sum of the link costs in the corresponding cost sequences (we assume sum $\{\} = \infty$). E_3 follows from invariance rule.

We define the following:

Consistent_distances. Boolean function. True iff (1) distance of any node pair equals path cost of its route, (2) distance of any node pair via a neighbor equals path cost of its route via that neighbor, and (3) any distance in transit in a message equals path cost of the route in the same message. Formally,

$= [\forall u, z \in NODES : Dist_u(z) = Path\_cost(Route_u(z))]$

$\wedge\ [\forall u, z \in NODES, \forall v \in neighbors(u) : Distvia_u(v,z) = Path\_cost(Routevia_u(v,z))]$

$\wedge\ [\forall (z,d,p,cs,rd) \text{ in transit} : d = Path\_cost(p)]$.

Done. Set of node pairs. Formally,

$= \{(u,z) \in Reachable : [\forall x \in Route_u(z) : Has\_optimal\_path(x,z)\ \wedge\ [\forall v \in neighbors(x) : Channel_{xv}(z) = \langle\rangle]]\}$

The proof of N_1 is summarized in Figure 3. **A2** achieves N_1 in two stages: first within N steps $Consistent\_distances(z)$ is established; after that within H steps $Done = Reachable$ is established (which implies the right side of N_1). Formally,

(N_3) $Symmetric\ \wedge\ Step = 0$ *leads–to* $\mathcal{TC}\ \vee\ Step \leq N\ \wedge\ Consistent\_distances$

(N_4) $Symmetric\ \wedge\ Step = j\ \wedge\ Consistent\_distances$

leads–to $\mathcal{TC}\ \vee\ Step \leq j + H\ \wedge\ Done = Reachable$

N_1 follows from N_3, N_4 and C_1 by closure. N_2 follows from N_3 by closure. Thus it suffices to prove N_3 and N_4.

Proof of N_3

We define the following:

$k\_Consistent(\langle x_0, \ldots, x_n\rangle, \langle c_0, \ldots, c_n\rangle)$. Boolean function where $\langle x_0, \ldots, x_n\rangle$ is a *ud-path* and c_i's are costs. True iff the link costs of the first k links (x_i, x_{i+1}) equal respectively the first k costs c_i. Formally,

$= true$ iff for $i \in [0, .., \min(k-1, n-1)] : c_i = Linkcost_{x_i}(x_{i+1})$.

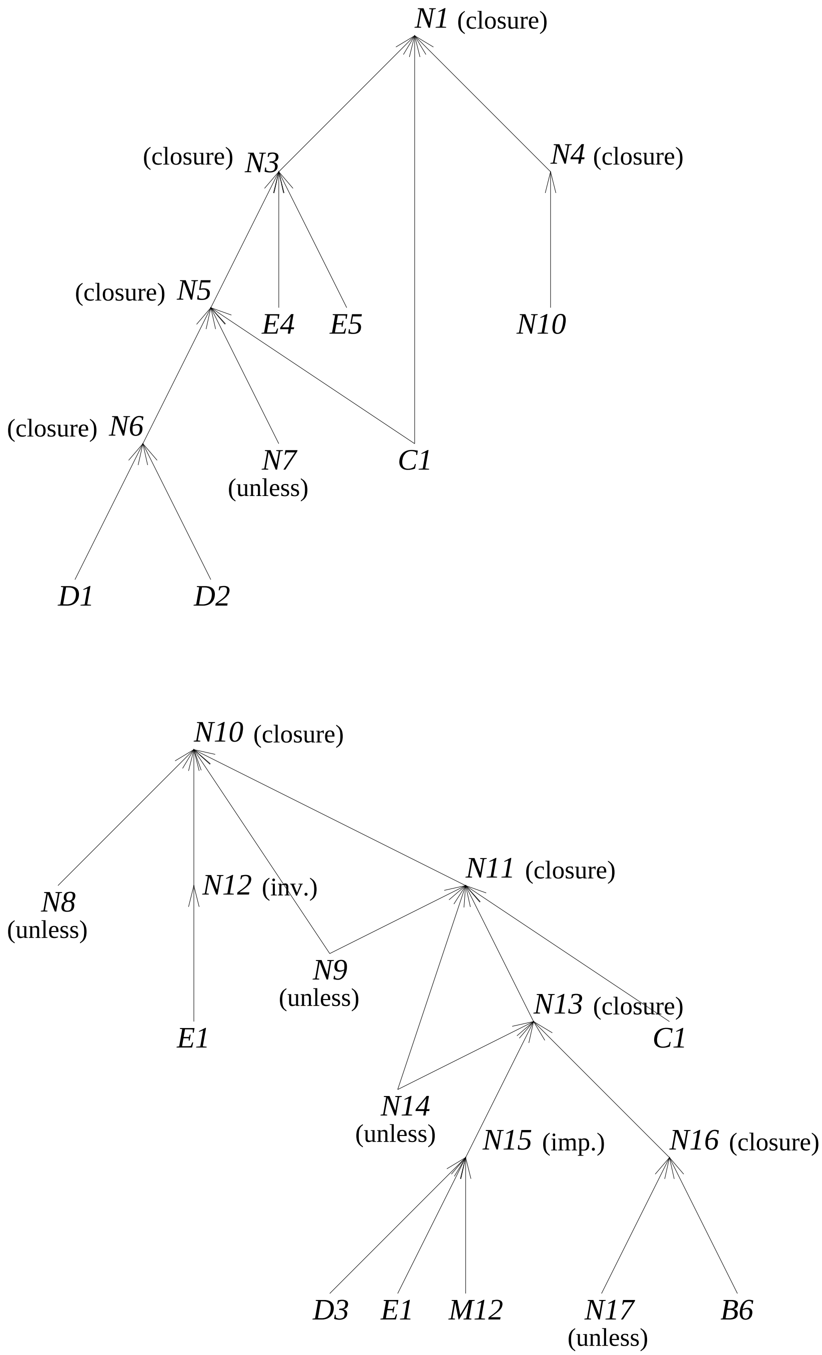

Figure 3: Proof of $N1$.

$k\_Consistent\_Routes$. Boolean function. True iff all routes are k–consistent. Formally,

$= [\forall u, z \in NODES : k\_Consistent(Route_u(z), Costseq_u(z))]$

$\wedge\ [\forall u, z \in NODES, v \in neighbors(u) : k\_Consistent(Routevia_u(v,z), Costseqvia_u(v,z))]$

$\wedge\ [\forall (z, d, p, cs, rd) \text{ in transit} : k\_Consistent(p, cs)]$.

Note that the first argument of $k\_Consistent$ in the definition of $k\_Consistent\_Routes$ is a *ud-path* (from E_1).

(N_5) $Symmetric \wedge Step = 0$ *leads–to* $\mathcal{TC} \vee Step \leq N \wedge N\_Consistent\_Routes$

N_5 states that within N steps all routes are N-consistent. N_3 follows from N_5, E_2 and E_3 by closure. (For the *Routevia* part of *Consistent_Distances*, the details are as follows: From E_3, $Distvia_u(v,z) = \text{sum } \{c : c \in Costseqvia_u(v,z)\}$. From $N\_Consistent\_Routes$ (right side of N_5) and $|Routevia_u(v,z)| \leq N$ (E_2), we have that $Distvia_u(v,z)$ equals the current cost of $Routevia_u(v,z)$.) Thus it suffices to prove N_5.

(N_6) $Symmetric \wedge Step \leq k \wedge k\_Consistent\_Routes$

leads–to $\mathcal{TC} \vee Step \leq k+1 \wedge (k+1)\_Consistent\_Routes$

(N_7) $Symmetric \wedge k\_Consistent\_Routes$ *unless* $\mathcal{TC}$

N_6 state that if at step k all routes are k–consistent, then within one more step all routes are $(k+1)$–consistent. N_7 state that once $k\_Consistent\_Routes$ is established, it continues to hold. N_7 follows from the unless rule. N_5 follows from N_6, N_7 and C_1 by closure (since 0–consistency is true for any route at any step).

Thus it suffices to prove N_6. N_6 follows from D_1, and D_2 by closure. To see this, suppose the route and the cost sequence of a node v are p and cs, respectively (p and cs are k–consistent). Then in at most one step, the route and the cost sequence of a neighbor u via v either become $\langle u\rangle@p$ and $\langle Linkcost_u(v)\rangle@cs$, or become $\langle\rangle$ and $\langle\rangle$. In either case, they are $(k+1)$–consistent.

Proof of N_4

We first define the following:

$depth(u,z)$. Minimum length (in number of links) of a minimum cost path from u to z. Formally,

$= \min\{|p| : p \in Availablepaths(u,z) \wedge Path\_cost(p) = Cost(u,z)\}$.

Note that $depth(z,z) = 0$ since $|\langle z\rangle| = 0$.

k_Reachable. Subset of node pairs in *Reachable* with depth less than or equal to k. Formally,

$= \{(u,z) \in Reachable : depth(u,z) \leq k\}$.

Some safety assertions:

(N_8) $depth(u,z) = k$ *unless* $\mathcal{TC}$

N_8 states that the value of $depth(u, z)$ does not change. N_8 follows from the unless rule.

(N_9) $Symmetric \wedge Consistent\_distances$ *unless* $\mathcal{TC}$

N_9 follows from the unless rule.

(N_{10}) $Symmetric \wedge Step = j \wedge Consistent\_distances$ *leads–to* $\mathcal{TC} \vee Step \leq j + k \wedge k\_Reachable \subseteq Done$

N_{10} states that once consistent distances are obtained, within k steps, $Done$ will contain all nodes in $k\_Reachable$. N_4 follows from N_{10} by replacing k by H (note that $H\_Reachable = Reachable$ from the definition of H). Thus it suffices to prove N_{10}.

(N_{11}) $Symmetric \wedge Step = j \wedge Consistent\_distances \wedge k\_Reachable \subseteq Done$
leads–to $\mathcal{TC} \vee Step \leq j + 1 \wedge (k+1)\_Reachable \subseteq Done$

(N_{12}) Invariant $Step > 0 \Rightarrow (z, z) \in Done$

N_{11} states that once consistent distances are obtained and $Done$ contains all nodes in $k\_Reachable$, within one step $Done$ will contain all nodes in $(k+1)\_Reachable$. N_{12} states that $Done$ includes $0\_Reachable$ after all messages generated by topology change events are received (at this time, outgoing channels of z do not contain any message for destination z). N_{10} follows from N_9, N_{11}, N_{12}, and N_8 by closure. N_{12} follows from E_1 using the invariance rule (from E_1, a message received by z does not contain a distance for z, hence z always has an optimal path).

(N_{13}) $Symmetric \wedge Step = j \wedge Consistent\_distances \wedge k\_Reachable \subseteq Done \wedge depth(u, z) = k + 1$
leads–to $\mathcal{TC} \vee Step \leq j + 1 \wedge (u, z) \in Done$

(N_{14}) $Symmetric \wedge Consistent\_distances \wedge \mathrm{S} \subseteq Done$ *unless* $\mathcal{TC}$

N_{13} states that once consistent distances are obtained and $Done$ contains all nodes in $k\_Reachable$, within one step a node u in $Reachable$ with depth $k+1$ will join $Done$. N_{14} states that once consistent distances are obtained, $Done$ does not shrink. N_{14} follows from the unless rule. N_{11} follows from N_9, N_{13}, N_{14}, and C_1 by closure. Thus it suffices to prove N_{13}.

(N_{15}) Invariant $Symmetric \wedge Consistent\_distances \wedge k\_Reachable \subseteq Done \wedge depth(u, z) = k + 1$
$\Rightarrow Has\_Optimal\_Path(u, z)$

(N_{16}) $Symmetric \wedge Step = j \wedge Consistent\_distances \wedge k\_Reachable \subseteq Done \wedge depth(u, z) \leq k + 1$
leads–to $\mathcal{TC} \vee Step \leq j + 1 \wedge [\forall v : Channel_{uv}(z) = \{\}]$

N_{15} states that if consistent distances are obtained and $Done$ contains all nodes in $k\_Reachable$, a node at depth $k+1$ has an optimal next–hop path. N_{16} states that once consistent distances are obtained and

$Done$ contains all nodes in $k\_Reachable$, within one step, outgoing channels of a node pair (u, z) at depth $k+1$ will not contain any messages for z.

N_{13} follows from N_{15}, N_{16}, and N_{14} by closure.

N_{15} follows from D_3, E_1 and M_{12} by the implication. (The details are as follows: Consider (u, z) satisfying the left side of N_{15}. Consider v, a next node on a shortest length optimal path from u to z. From the definition of $depth$, $depth(v, z) = k$. From the left side of N_{15}, the outgoing channels of v do not contain any messages for z. Hence, from D_3, $Routevia_u(v, z)$ is an optimal path. From $Consistent\_distances$ (left side of N_{15}) and E_1, distances via all other neighbors of u equal cost of some path. Hence, from M_{12}, u has an optimal path.) Thus it suffices to prove N_{16}.

$$(N_{17})\ \ Symmetric\ \wedge\ Consistent\_distances\ \wedge\ k\_Reachable \subseteq Done\ \wedge\ depth(u,z) \leq k+1$$
$$\wedge\ \text{sum}\ \{|Channel_{uv}(z)| : v \in neighbors(z)\} \leq n\ \ \textit{unless}\ \ \mathcal{TC}$$

N_{17} states that once $Consistent\_distances$ is achieved, and $Done$ contains $k\_Reachable$, the number of messages in the outgoing channels of a node u at depth $k+1$ does not increase.

N_{16} follows from N_{17} and B_6 by closure. N_{17} follows from the unless rule. This completes the proof of the theorem.

End of proof of Theorem 3

5 Algorithm A3

Table 3 specifies the state variables and events of a node in **A3**. (Refer to the table in the following discussion.) **A3** differs from **A2** only in the procedure $Update\&Send$.

In **A3**, the node id's are considered totally ordered. Node u chooses a neighbor v as its next–hop for destination z iff (i) v is the minimum node in $Best\_hops_u(z)$, and (ii) v is the minimum node in $Best\_hops_u(x)$ for every node x in the route to z via v. If there is no such v, then the next–hop is nil. (See definition of $Min\_best\_hop(z)$ in the table.)

Procedure $Update\&Send$ considers a destination z as affected if (1) distance for z has changed, or (2) route for z has changed, or (3) some node x on $Route_u(z)$ is affected. This ensures that if the next–hop changes for a destination x, which is on the route to another destination z, the next–hop for z also changes.

This way of choosing next–hops and affected destinations ensures that during convergence (when the routes are not stable), the following property P holds: the next–hop of u for destination z is also the next–hop for all intermediate destinations on $Route_u(z)$.

Note that in **A3**, node u may choose the next–hop for destination z to be nil, when in fact there is a neighbor v, and chosing v as the next-hop to z satisfies P. Although it may seem that this slows down the convergence, there is a good reason for doing this: if the minimum node in $Best\_hops_u(z)$, say w, does not satisfy P, then it means that u has inconsistent distances via v and w.

Theorem 4. **A3** *satisfies* M_1 *and* M_2.

Proof of Theorem 4

Proof of Theorem 4 is identical to proof of Theorem 2 with the following changes. We redefine In and $Lowest$ for **A3** as follows:

In. Maximal subset of $Reachable$ such that (u, z) is a member of In iff

(1) $Has\_optimal\_path(u, z)$,

(2) for any message (x, d, p, cs, rd) in transit, $Dist_u(z)$ is less than rd,

(3) for any node pair (w, x) in $Reachable$ not satisfying $Has\_optimal\_path(w, x)$,

$Dist_u(z) < \min\{Distvia_w(v, x) : v \in neighbors(w)\}$ and $Dist_u(z) < Cost(w, x)$.

$Lowest$. Formally,

$$= \min(\{Cost(x, z) : (x, z) \in Out\},$$
$$\{Distvia_u(v, z) : (u, z) \in Out \ \wedge \ v \in neighbors(u)\},$$
$$\{rd : (x, d, p, cs, rd) \in Channel_{uv} \ \wedge \ (u, x), (v, x) \in Reachable\}).$$

We modify the assertions M_{12} and M_{14} as follows:

(M_{12}) Invariant $u \neq z \ \wedge \ Dist_u(z) \neq \infty \ \Rightarrow \ Dist_u(z) = \min\{Distvia_u(v, z) : v \in neighbors(u)\}$

(M_{14}) Invariant $Cost(u, z) \leq Lowest \ \wedge \ u \neq z$

$$\Rightarrow [\exists (v, z) \in In : Distvia_u(v, z) = Cost(u, z)] \ \wedge$$
$$[\forall (v, z) \in In : Distvia_u(v, z) = Cost(u, z) \ \wedge \ x \in Routevia_u(v, z) \ \Rightarrow \ Distvia_u(v, x) = Cost(u, x)]$$

M_{12} follows from invariance rule. M_{14} follows from B_2 and B_3 by implication.

Other assertions that hold for **A2** also hold for **A3**. Their proofs are identical except M_{11} which now follows from M_{20}, M_{21}, M_{12}, M_{13} and M_{14} by implication where M_{20} and M_{21} are as follows:

(M_{20}) Invariant $[\exists v \in neighbors(u) : [\forall x \in Routevia_u(v, z) : v = \min Best\_hops_u(x)]] \ \Rightarrow \ Dist_u(z) \neq \infty$

(M_{21}) Invariant $Lowest \geq k \ \wedge \ |DVia(k)| = 0 \ \wedge \ Distancevia_u(v, z) = Cost(u, z) = k$

$$\wedge \ x \in Routevia_u(v, z) \ \wedge \ Distvia_u(w, x) = Cost(u, x)$$
$$\Rightarrow \ Distancevia_u(w, z) = Cost(u, z)$$

M_{20} follows from invariance rule. M_{21} follows from B_2 and B_3 by implication. Hence M_1 and M_2 hold for

A3.

End of proof of Theorem 4

Theorem 5. *Assuming synchronous execution,* **A3** *satisfies* N_1 *and* N_2*.*

Proof of Theorem 5

Proof of Theorem 5 is identical to proof of Theorem 3 with $depth(u, z)$ being redefined. In **A2**, $depth(u, z)$ stood for minimum length of an optimal path from u to z. Many such paths can exist and any of them can be chosen by u. In **A3**, only one of these optimal paths can be chosen by u; i.e. the path $p = \langle x_0, x_1, \ldots, x_n \rangle$ where $x_0 = u$, $x_n = z$ and x_1 is the minimum-id neighbor of the neighbors on the optimal paths to x_i, for $i = 1, \ldots, n$. (Note that this may not be the shortest length optimal path.) We redefine $depth(u, z)$ to handle this:

$depth(u, z)$. Length of the optimal path from u to z such that the next hop in this path is the minimum-id neighbor providing an optimal path for all intermediate nodes in this path. Formally,

$$= |p| \text{ such that } p \in Availablepaths(u, z) \ \wedge \ Path\_cost(p) = Cost(u, z) \ \wedge$$
$$[\forall x \in p : front(tail(p)) = \min\{front(tail(q)) : q \in Availablepaths(u, x) \ \wedge$$
$$Path\_cost(q) = Cost(u, x)\}].$$

All assertions that hold for **A2** also hold for **A3**. Their proofs are identical except N_{15} now follows from M_{20}, M_{12} (as defined in the proof of Theorem 4), E_1 and D_3 by implication. Hence N_1 and N_2 hold for **A3**.

End of proof of Theorem 5

6 Algorithm A4

Table 4 specifies the state variables and events of a node in **A4**. Each node u maintains the state variables of **A3**, except that $Routevia_u(v, z)$ and $Route_u(z)$ are now auxiliary. Instead, node u maintains new state variables $Pfnodevia_u(v, z)$ and $Pfnode_u(z)$. In $Pfnodevia_u(v, z)$, node u maintains an estimate of the *prefinal* node via neighbor v (i.e. the last node before z on the path to z via v). $Pfnodevia_u(v, z)$ is equal to *nil* if node u believes z cannot be reached via v. State variable $Pfnode_u(z)$ indicates the prefinal node via node u's next–hop.

The messages of **A4** are like the message of **A3**, except that they now contain prefinal node information, and the route information is auxiliary (i.e. not implemented).

The events of algorithm **A4** are like those of algorithm **A3**, with the following twist: each node in **A4** uses its prefinal nodes to construct *prefinal–routes*, which take the place of the routes in **A3**. Node

u constructs its prefinal–route via neighbor v for destination z, referred to as $Pfroutevia_u(v,z)$, as follows: Start with a sequence $\langle z\rangle$; add to the left of this sequence the prefinal node via v for the leftmost element of the sequence, until either (1) node u is added, or (2) the prefinal node is nil, or (3) a loop is established. We use $Pfroute_u(z)$ to refer to the prefinal–route for destination z via the next–hop. (Formal definitions of functions $Pfroutevia_u(v,z)$ and $Pfroute_u(z)$ are in the table).

Theorem 6.

(a) **A4** *satisfies M_1 and M_2*

(b) Assuming synchronous execution, **A4** *satisfies N_1 and N_2.*

Proof of Theorem 6

Because the variables of **A4** (both auxiliary and non-auxiliary) are a superset of the non-auxiliary variables of **A3** and their domains are the same, there is a natural (projection) mapping from the states of **A4** to the states of **A3**. For any state s of **A4**, let s' denote the corresponding state of **A3**. It is obvious that event e of **A4** is enabled in any state s iff the corresponding event e of **A3** is enabled in s'. We next show that event e of **A4** updates the variables of **A3** in the same way as the corresponding event e of **A3**; more precisely, if event e of **A4** has a transition (s,t), then the corresponding event e of **A3** has a transition (s',t'). For this, it is sufficient to establish that the prefinal-routes of **A4** simulate accurately the routes of **A3**. This is specified by the following assertion:

$$
\begin{aligned}
(R_1)\ \ \textit{Invariant}\ \ & \big([Routevia_u(v,z) = Pfroutevia_u(v,z)] \ \vee\ [Routevia_u(v,z) = \langle\rangle \ \wedge\ Pfroutevia_u(v,z) = \langle z\rangle]\big) \ \wedge \\
& \big([Route_u(z) = Pfroute_u(z)] \ \vee\ [Route_u(z) = \langle\rangle \ \wedge\ Pfroute_u(z) = \langle z\rangle]\big) \ \wedge \\
& \big((z,d,pfn,p) = Channel_{uv}[j] \ \Rightarrow\ [p = PfMroute_{uv}(j,z)] \ \vee\ [p = \langle\rangle \ \wedge\ PfMroute_{uv}(j,z) = \langle z\rangle]\big)
\end{aligned}
$$

where $Channel_{uv}[j]$ denotes the j-th message in $Channel_{uv}$ ($Channel_{uv}[0]$ is the front) and $PfMroute_{uv}(j,z)$ is defined as follows:

$PfMroute_{uv}(j,z)$. Sequence of nodes. $\langle s_0,\ldots,s_n\rangle$ where

(a) $s_n = z$,

(b) for all $i \in [0..n-1] : s_i = \begin{cases} Pfnodevia_v(u,s_{i+1}) & \text{if for all } k \le j,\ Channel_{uv}[k] \text{ does not contain } (s_{i+1},d,pfn,p) \\ pfn & \text{if for largest } k \le j,\ Channel_{uv}[k] \text{ contains } (s_{i+1},d,pfn,p) \end{cases}$

(c) for all $i \in [1..n-1] : s_i \notin \{s_{i+1},\ldots,s_n\}$, and

(d) $s_0 = u \ \vee\ Pfnode_u(s_0) = nil \ \vee\ s_0 \in \{s_1,\ldots,s_n\}$.

R_1 states that the prefinal–routes and the routes (which are auxiliary) agree. R_1 follows from invariance rule.

Given R_1, if event e of **A4** has transition (s, t), then the corresponding event e of **A3** has transition (s', t'). We have already established that e of **A4** is enabled whenever e of **A3** is enabled. We also have that the initial condition of **A4** imply the initial condition of **A3**. Thus, **A4** is a strongly well–formed refinement of **A3**; that is, **A4** satisfies any safety or progress properties of **A3**. This and Theorem 4 imply Theorem 5.
End of proof of Theorem 6

7 Concluding Remarks

The algorithms analyzed in this paper are representative of various internetworking distance–vector routing protocols. Distance–vector routing algorithms are difficult to understand. Most of their analyses in the literature is operational. In the course of our work, we discovered that they are often incomplete or inaccurate; for example, reference [17] considers only one or two link failures rather than an arbitrary succession of topology changes, to prove the properties of their algorithm; the routing table update procedure in [3] is inaccurate; the example in [3] to illustrate $O(N)$ convergence is wrong, etc. A stepwise assertional design, such as the one presented here, is effective at making it easier to understand these algorithms.

In our opinion, the major drawback of our stepwise design is that we could not obtain a refinement result for algorithm **A2** and **A3** similar to the result for algorithm **A4**. Instead, we had to check that the proof that **A1** eventually achieves optimal paths also holds for **A2** and **A3**, and that the proof that **A2** achieves optimal paths in $N + H$ steps also holds for **A3**.

References

[1] C. Alaettinoğlu and A. U. Shankar. Stepwise assertional design of distance–vector routing algorithms. In *IFIP Protocol Specification Testing and Verification '92*, Orlando, Florida, June 1992.

[2] K. M. Chandy and J. Misra. *Parallel Program Design*, chapter 2,3 and 8. Addison-Wesley, 1988.

[3] C. Cheng, R. Riley, S. P. R. Kumar, and J. J. Garcia-Luna-Aceves. A loop-free Bellman-Ford routing protocol without bouncing effect. In *ACM SIGCOMM '89*, pages 224–237, September 1989.

[4] E. Dijkstra and C. Scholten. Termination detection for diffusing computations. *Information Processing Letters*, 11(1):1–4, 1980.

[5] L. Ford and D. Fulkerson. *Flows in Networks*, pages 297–333. Prentice-Hall, Inc., 1962.

[6] J.J. Garcia-Luna-Aceves. A unified approach to loop free routing using distance vectors or link states. In *ACM SIGCOMM '89*, pages 212–223, September 1989.

[7] C. Hedrick. Routing information protocol. Request for Comment RFC-1058, Network Information Center, June 1988.

[8] C. L. Hedrick. An introduction to igrp. Technical report, The State University of New Jersey, Center for Computer and Information Services, Laboratory for Computer Science Research, Rutgers, New Jersey, August 1991. Available by anonymous ftp from `ftp.cisco.com`.

[9] P. A. Humblet. Another adaptive distributed shortest path algorithm. *IEEE Transactions on Communications*, 39(6):995–1003, June 1991.

[10] J. M. Jaffe and F.H. Moss. A responsive distributed routing algorithm for computer networks. *IEEE Transactions on Communications*, COM-30(7):1758–1762, July 1982.

[11] S. S. Lam and A. U. Shankar. A relational notation for state transition systems. *IEEE Transactions on Software Engineering*, SE-16(7):755–775, July 1990. An abriviated version appeared in *Protocol Specification, Testing, and Verification VIII*, 1988.

[12] K. Lougheed and Y. Rekhter. Border gateway protocol (bgp). Request for Comment RFC-1105, Network Information Center, June 1989.

[13] Z. Manna and A. Pnueli. Adequate Proof Principles for Invariance and Liveness Properties of Concurrent Programs. *Science of Computer Programming*, 4:257–288, 1984.

[14] J. M. McQuillan, G. Falk, and I. Richer. A review of the development and performance of the Arpanet routing algorithm. *IEEE Transactions on Communications*, COM-26:1802–1811, Dec 1978.

[15] P. M. Merlin and A. Segall. A failsafe distributed routing protocol. *IEEE Transactions on Communications*, COM-27(9):1280–1287, September 1979.

[16] S. Owicki and D. Gries. An axiomatic proof technique for parallel programs i. *Acta Informatica*, 6:319–340, 1976.

[17] B. Rajagopalan and M. Faiman. A new responsive distributed shortest-path routing algorithm. In *ACM SIGCOMM '89*, pages 237–246, September 1989.

[18] Y. Rekhter. Inter-domain routing protocol (idrp). Available from the author., 1992. T.J. Watson Research Center, IBM Corp.

[19] A. Segall. Advances in verifiable fail-safe routing procedures. *IEEE Transactions on Communications*, COM-29(4):491–497, April 1981.

[20] A.U. Shankar. An Introduction to Assertional Reasoning for Concurrent Systems. *ACM Computing Surveys*, September 1993. to appear.

[21] K. G. Shin and M. Chen. Performance analysis of distributed routing strategies free or ping-pong-type looping. *IEEE Transactions on Computers*, 1987.

Table 1: Algorithm **A1**.

State variables and initial conditions of node u:

$Linkcost_u(v) : I^+ \cup \{\infty\}$. Initially ∞. Cost of the link (u, v).

$Distvia_u(v, z) : I^+ \cup \{\infty\}$. Initially ∞. Distance to destination z via neighbor v.

$Nhop_u(z) : neighbors(u) \cup \{nil\}$. Initially nil. Next-hop for destination z.

$Dist_u(z) : I^+ \cup \{0, \infty\}$. Initially ∞ for $u \neq z$, and 0 for $u = z$. Distance to destination z via next–hop.

Events of node u:

$Receive_u(v, d\_vector)$
  action: $Update\&Send_u(v, \{(z, d + Linkcost_u(v)) : (z, d) \in d\_vector\})$ $\{d$ can be $\infty\}$

$LinkCostChange_u(v, newcost)$ $\{newcost \neq \infty\}$
  enabled: $Linkcost_u(v) < \infty$
  action: $c := newcost - Linkcost_u(v)$; $Linkcost_u(v) := newcost$;
    $Update\&Send_u(v, \{(z, Distvia_u(v, z) + c) : \forall z \in NODES\})$

$LinkFailure_u(v)$
  enabled: $Linkcost_u(v) < \infty$
  action: $Channel_{uv} := \langle\rangle$; $Linkcost_u(v) := \infty$;
    $Update\&Send_u(v, \{(z, \infty) : \forall z \in NODES\})$

$LinkRecovery_u(v, newcost)$ $\{newcost \neq \infty\}$
  enabled: $Linkcost_u(v) = \infty$
  action: $Linkcost_u(v) := newcost$;
    $Update\&Send_u(v, \{(v, newcost)\})$;
    Send$(u, \{(z, Dist_u(z)) : \forall z \in NODES\})$ to v

$Update\&Send_u(v, d\_vector)$
  local variable *affectedsinks* initially $\{\}$;
  for all $(z, d) \in d\_vector$ do
    $Distvia_u(v, z) := d$; {Note that d can be ∞}
    if $[Nhop_u(z) \neq v \ \wedge \ Distvia_u(v, z) < Dist_u(z)]$
      $\vee \ [Nhop_u(z) = v \ \wedge \ Distvia_u(v, z) \neq Dist_u(z)]$ then
      if $Best\_hops_u(z) \neq \{\}$ then
        for some $k \in Best\_hops_u(z)$ do
          $Nhop_u(z) := k$; $Dist_u(z) := Distvia_u(k, z)$
      else
        $Nhop_u(z) := nil$; $Dist_u(z) := \infty$
      endif;
      $affectedsinks := affectedsinks \cup \{z\}$
    endif
  for all w such that $(u, w) \in UPLINKS$ do
    Send$(u, \{(z, Dist_u(z)) : \forall z \in affectedsinks\})$ to w;

where $Best\_hops_u(z)$ is a function that returns the following subset of $neighbors(u)$:

$$\{v : Distvia_u(v, z) \neq \infty \ \wedge \ Distvia_u(v, z) = \min\{Distvia_u(w, z) : w \in neighbors(u)\} \ \}$$

Table 2: Algorithm **A2**.

State variables and initial conditions of node u:

$Linkcost_u(v), Nhop_u(z), Dist_u(z), Distvia_u(v,z)$. As in **A1**.

$Routevia_u(v,z)$. sequence of nodes. Initially $\langle\rangle$. Path from u to z via v.

$Costseqvia_u(v,z)$. sequence of $I^+ \cup \{\infty\}$. Auxiliary. Initially $\langle\rangle$. Sequence of link costs on $Routevia_u(v,z)$.

$Route_u(z)$. sequence of nodes. Initially $\langle\rangle$ for $u \neq z$, $\langle u\rangle$ for $u = z$. Path from u to z.

$Costseq_u(z)$. sequence of $I^+ \cup \{\infty\}$. Auxiliary. Initially $\langle\rangle$ for $u \neq z$, $\langle 0\rangle$ for $u = z$.
Sequence of link costs on $Route_u(z)$.

Events of node u:

$Receive_u(v, d\_vector)$
- action: local variable $d\_vector2$: initially $\{\}$;
  - $d\_vector2 := \{(z, d + Linkcost_u(v), \langle u\rangle @ p, \langle Linkcost_u(v)\rangle @ cs) : (z,d,p,cs,rd) \in d\_vector \wedge d \neq \infty\}$
  - $\cup\{(z, \infty, \langle\rangle, \langle\rangle) : (z,d,p,cs,rd) \in d\_vector \wedge d = \infty\}$;
  - $Update\&Send_u(v, d\_vector2)$

$LinkCostChange_u(v, newcost)$ $\{newcost \neq \infty\}$
- enabled: $Linkcost_u(v) < \infty$
- action: local variable d_vector : initially $\{\}$; c;
  - $c := newcost - Linkcost_u(v)$; $Linkcost_u(v) := newcost$;
  - $d\_vector := \{(z,\ Distvia_u(v,z) + c,\ Routevia_u(v,z),\ \langle newcost\rangle @ tail(Costseqvia_u(v,z))) : \forall z \in NODES\}$;
  - $Update\&Send_u(v, d\_vector)$

$LinkFailure_u(v)$
- enabled: $Linkcost_u(v) < \infty$
- action: $Channel_{uv} := \langle\rangle$; $Linkcost_u(v) := \infty$;
  - $Update\&Send_u(v, \{(z, \infty, \langle\rangle, \langle\rangle) : \forall z \in NODES\})$;

$LinkRecovery_u(v, newcost)$ $\{newcost \neq \infty\}$
- enabled: $Linkcost_u(v) = \infty$
- action: $Linkcost_u(v) := newcost$;
  - $Update\&Send_u(v, \{(v, newcost, \langle u, v\rangle, \langle Linkcost_u(v), 0\rangle)\})$;
  - Send$(u, \{(z, Dist_u(z), Route_u(z), Costseq_u(z), Dist_u(z)) : \forall z \in NODES \wedge v \notin Route_u(z)\}$
  - $\cup\{(z, \infty, \langle\rangle, \langle\rangle, Dist_u(z)) : \forall z \in NODES \wedge v \in Route_u(z)\})$ to v

$Update\&Send_u(v, d\_vector)$
- local variable $affectedsinks$: initially $\{\}$;
- for all $(z,d,p,cs) \in d\_vector$ do $\{$Note that d can be $\infty\}$
  - $Distvia_u(v,z) := d$; $Routevia_u(v,z) := p$; $Costseqvia_u(v,z) := cs$;
  - if $(Nhop_u(z) \neq v \wedge Distvia(z,v) < Dist_u(z))$
    - $\vee\ (Nhop_u(z) = v \wedge (Distvia_u(v,z) \neq Dist_u(z) \vee Routevia_u(v,z) \neq Route_u(z)))$ then
    - if $Best\_hops_u(z) \neq \{\}$ then
      - for some $k \in Best\_hops_u(z)$ do
        - $Nhop_u(z) := k$; $Dist_u(z) := Distvia_u(k,z)$;
        - $Route_u(z) := Routevia_u(k,z)$; $Costseq_u(z) := Costseqvia_u(k,z)$
    - else
      - $Nhop_u(z) := nil$; $Dist_u(z) := \infty$; $Route_u(z) := \langle\rangle$; $Costseq_u(z) := \langle\rangle$
    - endif;
    - $affectedsinks := affectedsinks \cup \{z\}$
  - endif
- for all w such that $(u,w) \in UPLINKS$ do
  - local variable d_vector : initially $\{\}$;
  - $d\_vector := \{(z, \infty, \langle\rangle, \langle\rangle, Dist_u(z)) : w \in Route_u(z) \wedge z \in affectedsinks\}$
  - $\cup\{(z, Dist_u(z), Route_u(z), Costseq_u(z), Dist_u(z)) : w \notin Route_u(z) \wedge z \in affectedsinks\}$;
  - Send(u, d_vector) to w;

where $Best\_hops_u(z)$ is as defined in **A1** (Table 1).

Table 3: Algorithm **A3**.

State variables and initial conditions of node u:

$Linkcost_u(v), Nhop_u(z), Dist_u(z), Distvia_u(v,z), Route_u(z), Routevia_u(v,z), Costseq_u(z), Costseqvia_u(v,z)$.
As in **A2**.

Events of node u:

$Receive_u, LinkCostChange_u, LinkFailure_u, LinkRecovery_u$. As in **A2**.

```
Update&Send_u(v, d_vector)
    local variable affectedsinks: initially {};
    for all (z, d, p, cs) ∈ d_vector do          {Note that d can be ∞}
        Distvia_u(v, z) := d; Routevia_u(v, z) := p; Costseqvia_u(v, z) := cs;
        if (Nhop_u(z) ≠ v ∧ Distvia_u(v, z) < Dist_u(z))
          ∨ (Nhop_u(z) = v ∧ (Distvia_u(v, z) ≠ Dist_u(z) ∨ Routevia_u(v, z) ≠ Route_u(z))) then
            affectedsinks := affectedsinks ∪ {z}
        endif
    for all z ∈ NODES−affectedsinks do
        if [∃x : x ∈ Route_u(z) ∧ x ∈ affectedsinks] then
            affectedsinks := affectedsinks ∪ {z}
        endif
    for all z ∈ affectedsinks do
        if Min_best_hop_u(z) ≠ {} then
            for some k ∈ Min_best_hops_u(z) do
                Nhop_u(z) := k; Dist_u(z) := Distvia_u(k, z);
                Route_u(z) := Routevia_u(k, z); Costseq_u(z) := Costseqvia_u(k, z)
        else
            Nhop_u(z) := nil; Dist_u(z) := ∞; Route_u(z) := ⟨⟩; Costseq_u(z) := ⟨⟩
        endif
    for all w such that (u, w) ∈ UPLINKS do
        local variable d_vector : initially {};
        d_vector :={(z, ∞, ⟨⟩, ⟨⟩, Dist_u(z)) : w ∈ Route_u(z) ∧ z ∈ affectedsinks}
                  ∪{(z, Dist_u(z), Route_u(z), Costseq_u(z), Dist_u(z)) : w ∉ Route_u(z) ∧ z ∈ affectedsinks};
        Send(u, d_vector) to w;
```

where the function $Min\_best\_hop_u(z)$ is now defined as follows:

$$\{v : [\forall x \in Routevia_u(v,z) : v = \min Best\_hops_u(x)]\}$$

where the function $Best\_hops_u(x)$ is as defined in **A1**.

Table 4: Algorithm **A4**.

State variables and initial conditions of node u:

$Linkcost_u(v), Nhop_u(z), Dist_u(z), Distvia_u(v,z)$. As in **A3**.

$Route_u(z), Routevia_u(v,z)$. Auxiliary. As in **A3**.

$Pfnodevia_u(v,z) : neighbors(z) \cup \{nil\}$. Initially nil. Prefinal node on the path from u to z via v.

$Pfnode_u(z) : neighbors(z) \cup \{nil\}$. Initially nil. Prefinal node on the path from u to z.

Functions:

$Pfroute_u(z)$: sequence of nodes. $\langle s_0, \ldots, s_n \rangle$ where
$s_n = z$,
for all $i \in [0..n-1] : s_i = Pfnode_u(s_{i+1})$,
for all $i \in [1..n-1] : s_i \notin \{s_{i+1}, \ldots, s_n\}$, and
$s_0 = u \ \vee \ Pfnode_u(s_0) = nil \ \vee \ s_0 \in \{s_1, \ldots, s_n\}$.

$Pfroutevia_u(v,z)$: Defined like $Pfroute_u(z)$ except $Pfnode_u(x)$ is replaced by $Pfnodevia_u(v,x)$.

Events of node u:

$Receive_u(v, d\_vector)$
action: local variable $d\_vector2$: initially $\{\}$;
$d\_vector2 := \{(z, \infty, nil, \langle\rangle) : (z,d,pfn,p) \in d\_vector \ \wedge \ d = \infty\}$
$\cup \{(z, d + Linkcost_u(v), u, \langle u \rangle @ p) : (z,d,pfn,p) \in d\_vector \ \wedge \ z = v \ \wedge \ d \neq \infty\}$
$\cup \{(z, d + Linkcost_u(v), pfn, \langle u \rangle @ p) : (z,d,pfn,p) \in d\_vector \ \wedge \ z \neq v \ \wedge \ d \neq \infty\}$;
$Update\&Send_u(v, d\_vector2)$

$LinkCostChange_u(v, newcost)$ $\{newcost \neq \infty\}$
enabled: $Linkcost_u(v) < \infty$
action: local variable d_vector : initially $\{\}$; c;
$c := newcost - Linkcost_u(v)$; $Linkcost_u(v) := newcost$;
$d\_vector := \{(z, Distvia_u(v,z) + c, Pfnodevia_u(v,z), Routevia_u(v,z)) : \forall z \in NODES\}$;
$Update\&Send_u(v, d\_vector)$

$LinkFailure_u(v)$
enabled: $Linkcost_u(v) < \infty$
action: $Channel_u(v) := \langle\rangle$; $Linkcost_u(v) := newcost$;
$Update\&Send_u(v, \{(z, \infty, nil, \langle\rangle) : \forall z \in NODES\})$

$LinkRecovery_u(v, newcost)$ $\{newcost \neq \infty\}$
enabled: $Linkcost_u(v) = \infty$
action: $Linkcost_u(v) := newcost$;
$Update\&Send_u(v, \{(v, newcost, u, \langle u, v \rangle)\})$
Send$(u, \{(z, \infty, nil, \langle\rangle) : v \in Pfroute_u(z) \ \wedge \ z \in affectedsinks\}$
$\cup \{(z, Dist_u(z), Pfnode_u(z), Route_u(z)) : v \notin Pfroute_u(z) \ \wedge \ z \in affectedsinks\})$ to v

Table 4 (cont.): Algorithm **A4**.

$Update\&Send_u(v, d\_vector)$
local variable *affectedsinks*: initially $\{\}$;
for all $(z, d, pfn, p) \in d\_vector$ do $\{$Note that d can be $\infty\}$
$Distvia_u(v,z) := d$; $Pfnodevia_u(v,z) := pfn$; $Routevia_u(v,z) := p$;
if $(Nhop_u(z) \neq v \ \wedge \ Distvia_u(v,z) < Dist_u(z))$
$\vee \ (Nhop_u(z) = v \ \wedge \ (Distvia_u(v,z) \neq Dist_u(z) \ \vee \ Pfroute_u(z) \neq Pfroutevia_u(v,z)))$ then
$affectedsinks := affectedsinks \cup \{z\}$
endif
for all $z \in NODES - affectedsinks$ do
if $[\exists k : k \in Pfroute_u(z) \ \wedge \ k \in affectedsinks]$ then
$affectedsinks := affectedsinks \cup \{z\}$
endif
for all $z \in affectedsinks$ do
if $Min\_best\_hop_u(z) \neq \{\}$ then
for some $k \in Min\_best\_hops_u(z)$ do
$Nhop_u(z) := k$; $Dist_u(z) := Distvia_u(k,z)$;
$Pfnode_u(z) := Pfnodevia_u(k,z)$; $Route_u(z) := Routevia_u(k,z)$;
else
$Nhop_u(z) := nil$; $Dist_u(z) := \infty$; $Pfnode_u(z) := nil$; $Route_u(z) := \langle\rangle$;
endif
for all w such that $(u,w) \in UPLINKS$ do
local variable d_vector : initially $\{\}$;
$d\_vector := \{(z, \infty, nil, \langle\rangle) : w \in Pfroute_u(z) \ \wedge \ z \in affectedsinks\}$
$\cup \{(z, Dist_u(z), Pfnode_u(z), Route_u(z)) : w \notin Pfroute_u(z) \ \wedge \ z \in affectedsinks\}$;
Send(u, d_vector) to w;

where the function $Min\_best\_hop_u(z)$ is now defined as follows:

$$\{v : [\forall x \in Pfroutevia_u(v,z) : v = \min Best\_hops_u(x)]\}$$

where the function $Best\_hops_u(x)$ is as defined in **A1**.