A New Protocol for Route Discovery
In Multiple-Ring Networks:
Part II – Multicast, Recovery and High-Speed Processing

by
R. Cohen and A. Segall

Technical Report #728
April 1992
A New Protocol For Route Discovery In Multiple-Ring Networks:  
Part II — Multicast, Recovery and High-Speed Processing

Reuven Cohen & Adrian Segall
Dept. of Computer Science
Technion IIT
Haifa 32000
Israel

Abstract

A new MAC protocol for route discovery in multiple-ring network, called the ERD-protocol, has been presented in Part I. The present paper introduces some extensions of this protocol. The first extension indicates how the ERD-protocol can support multicast source routing, where a station seeks routes to multiple destinations. The ERD-protocol supplies the source with a description of a collection of routes. The route collection forms a tree that spans the source and the destination group. This leads to an important advantage of source routing multicasting over transparent spanning tree multicasting, since multicast data-frames are sent on part of the network rather than on the entire network. Another addition presents a recovery mechanism from transmission errors and station failures. Finally it is shown how the ERD-protocol can be applied in networks with very high speed rings, where bridges do not have sufficient time, after recognizing the route identity, to look in the local table and decide whether the relevant fields in the received frame need to be altered.
1 Introduction

Route discovery protocol is a mechanism used by a source station before sending data-frames to its destination in a Source Routing network. In Part I [1], a new protocol for route discovery in multiple-ring networks, called the ERD-protocol (Efficient Route Discovery protocol) has been presented. The present paper introduces three extensions of this protocol.

Section 2 shows how the new protocol can support multicast source routing in a multiple-ring network. A source station that executes the protocol is provided with a sequence of bridge and ring identities that forms a tree-description. Each data-frame sent by the source, whose routing information field contains the tree-description, is ensured to be received by every member of the destination group exactly once. Moreover, the tree selected by the ERD-protocol spans only the relevant source and destinations, and multicast frames are sent only on this tree. This provides in most cases a significant improvement over transparent spanning tree multicast [2], where each data-frame is sent on every LAN in the network. An important feature is that the route discovery version of the ERD-protocol, as presented in Part I, is simply a special case of the tree discovery version. Thus, the source, bridges and destinations need not distinguish between the versions: the algorithms of the tree discovery version will reduce to route discovery where appropriate.

Section 3 introduces a recovery mechanism from error and failure conditions for the ERD-protocol. The recovery mechanism is based on the route discovery protocol suggested in [4]. Although this protocol uses a number of frames that can be exponential in the number of rings and bridges, the recovery mechanism is invoked unfrequently, so that its high communication cost does not significantly affect the load in the multiple-ring network.

One restriction of the basic protocol is that it requires bridges to make an on-line search in their local tables while repeating a search or a response frame, in order to determine whether certain bits in these frames must be changed. When the transmission rate is very high, it might be difficult to perform the search on-line. Section 4 modifies the ERD-protocol to allow bridges to perform the table search off-line, namely after having transmitted the received frame. The penalty of this version is that a bridge may need to send a response frame on its secondary ring, in addition to the response frame it sends on its primary ring in the basic protocol.

2 Multicast Source Routing

In the present section we investigate the extension of the ERD-protocol to support multicast source routing in the multiple-ring network. The difficulty with multicast source routing is to ensure that each destination will receive each data-frame sent by the source exactly once, while preventing the transmission of unnecessary copies of the frame in the network. The protocol presented in this section ensures these requirements.

The basic ERD-protocol finds a route description list from a source to one destination in multiple-ring network. As discussed in [1], the main advantage of the new protocol over previous protocols for the same purpose is its communication efficiency. Another important property of
the ERD-protocol is that it can be easily extended to find the description list of a tree to support routing from a source to a group of destinations. A tree-description is a collection of routes that form a tree whose root is the source station, and spans the entire destination group. Each frame sent by the source containing an appropriate tree-description in the routing information field is guaranteed to be received by each member of the destination group exactly once.

By giving the source stations the option to set a tree-description rather than a route-description in the routing information field, source-routing is extended to support multicast in addition to point-to-point communication. This extension is transparent to the extended-LAN bridges: upon receiving a source-routing frame, a bridge takes the same actions whether the routing information field of the frame contains a route-description or a tree-description. In fact, the distinction between a route-description and a tree-description is not necessary, since a route-description can be considered as a special case of a tree-description, spanning a single-destination group. As shown below, the distinction is not necessary in the route discovery protocol either: a bridge executes the same algorithm whether the protocol is supposed to find a route to a single destination or a tree to multiple destinations. The communication cost, in terms of transmitted frames, is also similar.

Using the new protocol for multicasting in a multiple-ring network leads to an important advantage over transparent spanning tree multicasting. This is because in a transparent spanning tree network, multicast is performed by sending a copy of the data-frame on every LAN [2], whereas the ERD-protocol ensures that in a source routing network, a copy of the data-frame is sent only on a tree spanning the relevant source and destinations.

In order to illustrate the problems associated with multicast source routing, consider the multiple-ring network of Fig. 1. Suppose that station s in ring R₁ wants to send a frame to a group $G = \{d₁, d₂, d₃\}$. Each data-frame sent by s to G contains the identity of s in the source field and the group identity G in the destination field. In addition, each data-frame has a routing information field that describes the route(s) from s to G. The contents of this field and the bridge
A possible way for \( s \) to determine the *routing information field* properly is to find a route to each member of \( G \) using a separate *route discovery protocol* and then to compose the three *route-descriptions* into a *tree-description*. However, there are two problems with this solution. The first problem is that in many cases the source station does not know the exact composition of the destination group, namely \( s \) may not know that \( G = \{d_1,d_2,d_3\} \); only the destinations know that they belong to the group. The second problem, arising when the destination is a large group, is that a separate execution of a route discovery protocol for each member of the destination group may take a long time and may require a large communication bandwidth, although the execution of a single *ERD-protocol* has very reasonable bandwidth requirements.

Since the *ERD-protocol* sends exactly one *SEARCH* frame on every ring, and bridges that send these *SEARCH* frames form, together with the corresponding rings, a tree rooted at the source, it is natural to extend it to find a *tree-description* for multicast. In this way, the abovementioned problems are overcome because:

- the source is not required to know the composition of the destination group.
- the number of frames required for the *route discovery protocol* does not depend on the size of \( G \), and is exactly the number of frames required to determine a route to a single destination, namely the number of rings plus the number of bridges in the multiple-ring network.

The multiple-ring network model considered from now on is exactly as the one described in Section 1 of Part I [1]. For simplicity it is assumed that a bridge connects exactly 2 token-rings. The extension to multi-port bridges is easy (see Section 5 in Part I).

### 2.1 Multicasting and Tree-Descriptions

In order to explain the meaning of a *tree-description* list, and how data-frames with such a list in the *routing information field* are routed in the source routing multiple-ring network, some definitions are needed. In the following, we consider a single source station \( s \) and a group of destinations \( G \). It is assumed that the source and the destinations are non-bridge stations.

**Definition 1:** a *route-designator* is a triple \( \{R_1, b, R_2\} \), where \( b \) is an identity of a bridge and \( R_1 \) and \( R_2 \) are the identities of the two rings connected by \( b \). The rings \( R_1 \) and \( R_2 \) are referred to as the *left* and the *right* rings of the route-designator; \( b \) is called the *bridge* of the route-designator.
Consider first the ring of the source

Claim

(i) destination receives the data-frame exactly once.

(ii) destination in G receives the data-frame more than once; therefore,

Proof

Consider a data-frame sent by the source algorithm performed by the source routing bridge

Lemma 1

{R 1 , b 2 , R 2 }, {R 2 , b 2 , R a } then copy the frame into a local buffer and transmit it on ring R 2 .

Note: If the right ring and the left ring of two consecutive route-designators are the same, one of them may be omitted. In addition, the route-designator braces may be omitted. Thus, for example, {R 1 , b 1 , R 2 , b 2 , R 3 , R 1 , b 3 , R 4 } is a tree-description that consists of the following route-designators: {R 1 , b 1 , R 2 }, {R 2 , b 2 , R 3 } and {R 1 , b 3 , R 4 }.

A source routing bridge is able to distinguish between a bridge identity and a source identity, and therefore it can extract the route-designators from the routing information field. The routing algorithm performed by the source routing bridge b that connects rings R 1 and R 2 , upon receiving a data-frame on one of its rings, say R 1 , is as follows: if the routing-information field contains the route-designator {R 1 , b, R 2 } then copy the frame into a local buffer and transmit it on ring R 2 .

Lemma 1

Consider a data-frame sent by the source s, that carries a tree-description from s to G. Then, the following hold:

(i) the data-frame is received by each member of the destination group G.

(ii) no bridge sends the data-frame unnecessarily; namely, a data-frame is sent on a ring R at most once, and only if R includes either a destination in G or a bridge that eventually transfers the frame from R to its other ring.

Note: Since a data-frame travels exactly one round-trip in a ring, (ii) above implies that no destination in G receives the data-frame more than once; therefore, (i) and (ii) imply that each destination receives the data-frame exactly once.

Proof

Claim (i) follows directly from part (a) of Definition 3 and from the source-routing bridge algorithm.

We start proving (ii) by showing that the data-frame is sent no more than once on any ring. Consider first the ring of the source R s. The data-frame is sent on R s by the source s, and since
from part (b) of Definition 3, ring \( R_s \) cannot be the right ring in any route-designator, no bridge sends the frame on \( R_s \). Therefore, the frame is sent exactly once on \( R_s \). In order to prove this for every other ring, assume that the data-frame is sent more than once on some ring \( R \neq R_s \). Consider a partition of the set of the network rings \( R \) into three sets: the set \( R_1 \) includes all rings on which the data-frame is sent only once, the set \( R_2 \) includes all rings on which the data-frame is sent more than once and the set \( R_0 \) includes all rings on which the data-frame is not sent at all.

The set \( R_1 \) is nonempty since it contains ring \( R_s \), and the set \( R_2 \) is nonempty since it contains ring \( R \). The only way for a data-frame to be sent on any ring except \( R_s \) is to be transferred by a bridge between its two rings. Thus the sets \( R_1 \) and \( R_2 \) cannot be disconnected from each other. Therefore, there must be at least one bridge, say \( b \), that receives the data-frame on a ring \( R_1 \in R_1 \), and sends it on a ring \( R_2 \in R_2 \). This implies that the tree-description in the routing information field contains the route-designator \( \{ R_1, b, R_2 \} \). From part (b) of Definition 3 follows that \( R_2 \) is the right ring of no other route-designator. Therefore, it must be bridge \( b \) that sends the data-frame more than once on \( R_2 \). This implies that bridge \( b \) receives the data-frame more than once on \( R_1 \). Since any data-frame travels exactly one round-trip in a ring, this means that the data-frame is sent more than once on \( R_1 \), contradicting the fact that \( R_1 \in R_1 \).

In order to complete the proof, we should show that if a data-frame is sent on a ring \( R, R \) includes either a destination in \( G \) or a bridge that eventually transfers the frame from \( R \) to its other ring. Suppose that bridge \( b \) recognizes a data-frame on ring \( R_1 \) and sends it on its other ring \( R_2 \). This implies that the tree-description recognizes a data-frame on ring \( R_1 \) and sends it on its other ring \( R_2 \). From part (c) of Definition 3 follows that the route-designator \( \{ R_1, b, R_2 \} \) belongs to some route-description from \( s \) to a member of the destination group \( G \). This means that either \( R_2 \) contains a member of \( G \), or the tree-description contains the route-designator \( \{ R_2, b', R_3 \} \), which implies that \( b' \) will send on \( R_3 \) the frame sent by bridge \( b \) on \( R_2 \).

Given a collection of route-designators, it is possible to build a directed graph \( (V, E) \) whose set of nodes \( V \) is the set of all left and right rings, and there is a directed edge from \( R_1 \in V \) to \( R_2 \in V \) if \( \{ R_1, b, R_2 \} \) is a route-designator in the collection. Now, it is possible to formulate requirements (a)-(c) of Definition 3 on this graph as follows:

**Definition 3**:

(a) each ring \( R \) that contains a station from \( G \) is represented by a node in \( V \).

(b) the directed graph is a directed tree rooted in the ring \( R_s \) containing the source, where edges are directed from the root towards the leaves.

(c) each leaf \( R \) in the tree represents a ring that contains a destination from \( G \).

It is easy to see that Definition 3 and 3* are equivalent.

The graph induced by a tree-description from \( s \) to \( G \) is called a routing-tree from \( s \) to \( G \).

**Note**: the routing-tree should not be confused with the feedback-tree used in the proofs of Part I.
Recall the example of Fig. 1, where station \( s \) in ring \( R_1 \) wants to multicast data-frames to group \( G \), where \( G \equiv \{d_1, d_2, d_3\} \). There are several possible ways to route data-frames from \( s \) to \( G \) such that each member of \( G \) will receive each data-frame exactly once and no bridge sends the data-frame unnecessarily, as shown in Fig. 2.

2.2 The Tree Discovery Version of The ERD-protocol

As mentioned above, the tree discovery protocol is exactly the basic ERD-protocol with some small modifications. The transmission and receipt of search and response frames is the same, i.e. according to Tables 1(a) and 2(a) of Part I. Tables 1(b), 2(b) and 3 are slightly changed, and the new ones appear here. The changes compared to the same tables is Part I are in boldface. Recall that the operator || denotes concatenation of two sequences. Therefore, the operation \( \text{list}(b) \leftarrow \text{list}(b)||\text{list} \) indicates that \( \text{list} \) is appended to the tail of \( \text{list}(b) \); e.g. if \( \text{list} = \{R_1, b_1, R_2, b_2, R_3\} \) and \( \text{list}(b) = \{R_4, b_4, R_5\} \), then after performing \( \text{list}(b) \leftarrow \text{list}(b)||\text{list} \), the sequence in \( \text{list}(b) \) is \( \{R_4, b_4, R_5, R_1, b_1, R_2, b_2, R_3\} \).

In the tree-description protocol, a destination in group \( G \) that recognizes a search frame that looks for the members of \( G \), repeats the frame with \( r_2 = 1 \), independently of the received value of \( r_2 \). The variable \( \text{list}(b) \) of every bridge \( b \) is built such that if \( R \) is a node in the routing-tree, the collection of route-designators in the list field of the response frames sent on \( R \) describe the subtree rooted at \( R \). This implies that the route-designators in the list field of the response frames sent on \( R_a \) describe the entire routing-tree. For example, in the protocol that finds the routing-tree of Fig. 2(c), the list fields of the response frames sent on ring \( R_2 \) are \( \{R_2, b_5, R_4\} \) and \( \{R_2, b_3, R_3\} \).

\(^1\)From now on we shall refer to these tables as Table 1(a) and Table 2(a).
upon leaving state 0: \[ \text{list}(b) \leftarrow \phi \]
upon leaving states 3 and 4: if \( b \) receives back \( \text{SEARCH}(1, r_2) \) then, \( \text{list}(b) \leftarrow \{R'(b), b, R''(b)\} \)
in states 5 and 6: if \( b \) receives on \( R'' \) \( \text{RESPONSE}(r_1, \text{list}) \) where \( \text{list} \neq \phi \) then

\[
\begin{align*}
\text{if list}(b) = \phi & \text{ then list}(b) \leftarrow \{R'(b), b\} || \text{list} \\
\text{else list}(b) & \leftarrow \text{list}(b) || \text{list}
\end{align*}
\]

Table 1(b): Additional Actions, Required for the Manipulation of \( \text{list}(b) \) in Bridge \( b \)

upon leaving state 0: \[ \text{list}(s) \leftarrow \phi \]
upon leaving state 4: if \( s \) receives back \( \text{SEARCH}(1, r_2) \), then members of \( G \) exist in the local ring
in state 6: if \( s \) receives \( \text{RESPONSE}(r_1, \text{list}) \), where \( \text{list} \neq \phi \) then \( \text{list}(s) \leftarrow \text{list}(s) || \text{list} \)

Table 2(b): Additional Actions Taken by the Source \( s \)

<table>
<thead>
<tr>
<th>State</th>
<th>Event</th>
<th>Reaction</th>
<th>Next State</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>receive ( \text{SEARCH}(r_1, r_2) ) where ( \text{my.id} \in G )</td>
<td>repeat ( \text{SEARCH}(r_1, 1) )</td>
<td>0</td>
</tr>
</tbody>
</table>

Table 3: Non-Source Algorithm

The concatenation of these sequences, \( \{R_2, b_5, R_4, R_2, b_3, R_3\} \) or \( \{R_2, b_3, R_3, R_2, b_5, R_4\} \), is the tree-description of the subtree rooted at \( R_2 \) of the tree shown in Fig. 2(c), whose sets of nodes and edges are \( \{R_2, R_3, R_4\} \) \( \{R_2 \xrightarrow{b_3} R_3, R_2 \xrightarrow{b_5} R_4\} \) respectively.

An important feature is that the route discovery version of the ERD-protocol, as presented in Part I, is simply a special case of the tree discovery version. Thus, the source, bridges and destinations need not distinguish between the two, but should simply perform the algorithms of the tree discovery version. When the destination is a single station, rather than a group, for every bridge \( b \) that receives a \( \text{RESPONSE} \) with \( \text{list} \neq \phi \) in state 5 or 6, the relation \( \text{list}(b) = \phi \) holds, and thus the algorithm in Table 1(b) here reduces to the corresponding algorithm in Part I. Similarly in the source algorithm, when \( s \) receives a \( \text{RESPONSE} \) with \( \text{list} \neq \phi \), the relation \( \text{list}(b) = \phi \) must hold when the destination is a single station, and thus the operation \( \text{list}(s) \leftarrow \text{list}(s) || \text{list} \) (Table 2(b) here) is in fact \( \text{list}(s) \leftarrow \text{list} \), as in the route discovery version (Table 2(b) in Part I).

### 2.3 Correctness Proof

The Termination property and the Communication Cost property of the basic ERD-protocol (Theorems 1 and 3 respectively in Part I respectively) rely on Tables 1(a) and 2(a) only. Since Tables 1(a) and 2(a) have not changed, these properties are valid for the tree discovery version as well. The purpose of this subsection is to state and prove the Reliable Response property for the tree discovery version.

**Theorem 1 (Reliable Response)**

Suppose that source \( s \) triggers the tree discovery protocol in order to find a tree-description to group \( G \). Suppose also that \( \text{list}(s) \neq \phi \) holds upon protocol termination, and that subsequently \( \text{list}(s) \neq \phi \) holds.
appears as the *routing information field* of a data-frame sent by $s$. Then, the data-frame is received exactly once by every member of $G$, and no copy of this frame is unnecessarily sent on any ring. 

**Note:** The case where $\text{list}(s) = \phi$ holds upon protocol termination is considered later, at the end of the section.

**Proof**

From Lemma 1 follows that we only need to show that the route-designator appearing in $\text{list}(s)$ form a tree-description from $s$ to $G$. Since definitions 3 and 3* are equivalent, we prove that the route-designators in $\text{list}(s)$ induce a graph that satisfies requirements (a), (b) and (c) of Definition 3*.

We first prove several preliminary claims:

**Claim (i):** If $\text{list}(b) \neq \phi$ holds for some $b \in B$, then the first three identities of $\text{list}(b)$ form the route-designator $\{R'(b), b, R''(b)\}$ and the last identity in $\text{list}(b)$ is the identity of some ring.

**Proof:** By induction on the time. Consider the first time when any bridge $b$ changes its $\text{list}(b)$ from $\phi$, and let $b'$ be that bridge. This may happen only when $b'$ performs $\text{list}(b') \leftarrow \{R'(b'), b', R''(b')\}$ (states 3 and 4 in Table 1(b)), because all other changes in $\text{list}(b')$ require $b'$ to receive a RESPONSE with $\text{list} \neq \phi$ from another bridge $b''$, which would imply that $b''$ had changed its $\text{list}(b'')$ from $\phi$ earlier.

Suppose the claim holds for the $\text{list}$ of every bridge until time $t$, and let $b$ be the first bridge that changes its $\text{list}$ afterwards. There are several possible reasons for this change:

- $b$ performs $\text{list}(b) \leftarrow \{R'(b), b, R''(b)\}$ (states 3 and 4 in Table 1(b)). In this case the claim clearly holds.
- $b$ performs $\text{list}(b) \leftarrow \{R'(b), b\} || \text{list}$ upon receiving a RESPONSE sent by some other bridge, say $b'$, on $R'(b')$ which is also $R''(b)$ (states 5 and 6 in Table 1(b)). The $\text{list}$ of this RESPONSE is in fact $\text{list}(b')$. Therefore, according to the induction assumption, the first identity in this $\text{list}$ is $R'(b')$, namely $R''(b)$, and the last identity belongs to some ring, say $R$. Thus, in the new $\text{list}(b)$, the first three identities are $\{R'(b), b, R''(b)\}$ and the last identity is $R$.
- $b$ performs $\text{list}(b) \leftarrow \text{list}(b) || \text{list}$ (states 5 and 6 in Table 1(b)) upon receiving a RESPONSE sent by some other bridge $b'$. In such a case, the three left identities of $\text{list}(b')$ do not change, and according to the induction assumption they are $\{R'(b), b, R''(b)\}$. The last identity of the new $\text{list}(b)$ is the last identity of $\text{list}= \text{list}(b')$, and thus, according to the induction assumption, is a ring identity.

**Claim (ii):** Suppose that for some $v \in B \cup \{s\}$, $\text{list}(v)$ contains the route-designator $\{R_1, b, R_2\}$.

Then, the following hold: (a) $R'(v) = R_1$; (b) $R''(v) = R_2$; (c) $b$ has sent a SEARCH on $R''(v)$.

**Proof:** By induction on time. Consider the first time when some $\text{list}(v)$ changes from $\phi$. This may happen only when $v$ is a bridge that performs $\text{list}(v) \leftarrow \{R'(v), v, R''(v)\}$ after having received back the SEARCH it has sent on $R''(v)$ (states 3 and 4 in Table 1(b)). In this case, (a), (b) and (c) obviously hold. Suppose the claim holds until time $t$ for $\text{list}(v')$ of every $v' \in B \cup \{s\}$ and
suppose that \( list(v) \) is the first to change afterwards. There are several possible reasons for this change:

- \( v \) is a bridge that performs \( list(v) \leftarrow \{ R'(v), v, R''(v) \} \) (states 3 and 4 in Table 1(b)). This case is similar to the one considered in the induction assumption.
- \( v \) is a bridge that receives a RESPONSE with \( list \neq \phi \) on \( R''(v) \) in states 5 or 6 and performs \( list(v) \leftarrow \{ R'(v), v \}\| list \) (see states 5 and 6 in Table 1(b)). Let \( b \) be the sender of this RESPONSE, namely \( list = list(b) \), and let \( R \) be the first identity in \( list \). Only one route-designator in the created \( list(v) \) had been missing in \( list(b) \) and this is \( \{ R'(v), v, R \} \). Thus, (a), (b) and (c) need to be proven only for this route-designator. Part (a) is immediate. Part (b) follows since \( R = R'(b) = R''(v) \), where the first inequality follows from Claim (i) and the second from the fact that the RESPONSE sent by \( b \) on \( R'(b) \) is received by \( v \) on \( R''(v) \). Part (c) holds from the fact that only a bridge that sends a SEARCH on its \( R'' \) can enter states 5 or 6.
- \( v \) receives a RESPONSE with \( list \neq \phi \) and performs \( list(v) \leftarrow list(v)\| list \) (states 5 and 6 in Table 1(b), or state 6 in Table 1(b)). Let \( b \) be the sender of this RESPONSE. This means that \( list \) is in fact \( list(b) \). According to Claim (i), the last identity in the old \( list(v) \), as well as the first identity of \( list(b) \) are ring identities. This implies that no new route-designator is created following the concatenation of the old \( list(v) \) and \( list(b) \), namely, all route-designators in the new \( list(v) \) had already existed in either the old \( list(v) \) or in \( list(b) \) (or in both). Thus, the claim holds from the induction assumption.

Claim (iii): Suppose that \( list(b) \neq \phi \), where \( b \in B \). Then, every route-designator in \( list(b) \) will finally appear in \( list(s) \).

Proof: Consider the path from \( b \) to \( s \) in the feedback-tree \( T \) (the tree considered in Part I). Let \( b \equiv v_1 \), and starting from \( b \) traverse this path and denote all nodes whose incoming edge in the path is solid with \( v_2, v_3, \ldots, v_K \), \( K \geq 2 \), such that \( s \equiv v_K \). From Lemma 5(b) in Part I follows that node \( v_2 \) receives the RESPONSE sent by \( v_1 \) (bridge \( b \)) before it enters state 8. Therefore, \( list(v_2) \) contains the sequence in \( list(v_1) \). By induction, for every \( 1 \leq i \leq K \), \( list(v_i) \) contains the sequence of every \( list(v_j) \), \( j < i \). This implies that \( list(v_K) \), i.e. \( list(s) \), contains the sequence of \( list(b) \).

We now prove that the sequence of route-designators in \( list(s) \) induces upon protocol termination a graph \( (V, E) \) that satisfies requirements (a), (b) and (c) of Definition 3*; namely, it is a routing-tree from \( s \) to \( G \). Before the proof of each part (a), (b) and (c), the claim of that part is recalled:

(a) Each ring \( R \) that contains a station from \( G \) is represented by a node in \( V \).

Proof: Note that \( list(s) \neq \phi \) implies that \( R''(s) \) is a node in \( V \). This is because \( list(s) \neq \phi \) means that \( s \) has received at least one RESPONSE with \( list \neq \phi \), and from Claim (i) follows that the first identity in this \( list \) is \( R''(s) \). Therefore, in the rest of the proof we consider all other rings.
Let $R$ be such a ring that contains a destination from $G$. Recall Lemma 2 in Part I that says that exactly one \textsc{search} is sent on every ring. Let $b$ be the bridge that sends a \textsc{search} on $R$. Since $R$ contains a destination from $G$, bridge $b$ receives its \textsc{search} back with $r_2 = 1$. At that time, $b$ sets $\text{list}(b) \leftarrow \{R'(b), b, R\}$ (see Table 1(b)), and from Claim (iii) follows that this route-designator appears in $\text{list}(s)$ as well, so $R$ is a node in $\mathcal{V}$.

(b) $(\mathcal{V}, \mathcal{E})$ is a directed tree rooted in $R''(s)$ whose edges are directed from the root $s$ towards the leaves.

\textbf{Proof:} according to [3], it is sufficient to show that: (1) $R''(s)$ has no incoming edge; (2) each $v \in \mathcal{V} - \{R''(s)\}$ has exactly one incoming edge and (3) there is no undirected cycle in $G$.

(1) In order to prove that $R''(s)$ has no incoming edge, we show that a route-designator $\{R, b, R''(s)\}$ cannot be created. From Claim (ii) follows that the creation of such route-designator implies that bridge $b$ sends a \textsc{search} on the ring of $s$, $R''(s)$. However, only one \textsc{search} is sent on $R''(s)$, and the sender of this \textsc{search} is $s$.

(2) We first prove that each $R \in \mathcal{V}$ has at least one incoming edge and then that no $R \in \mathcal{V}$ has more than one incoming edge. Let $R \in \mathcal{V}$ be a ring with no incoming edge in $\mathcal{E}$. Since $R$ is a node in $\mathcal{V}$, it must have an outgoing edge. Let this edge be $R \xrightarrow{b} R_1$. This implies that $\text{list}(s)$ contains the route-designator $\{R, b, R_1\}$. This route-designator is created by bridge $b$ and appears as the first 3 identities in $\text{list}(b)$, when $b$ sends a $\text{response}(\text{list}(b))$ on $R'(b)$. Let $b'$ be the bridge that sends a \textsc{search} on $R = R'(b)$. Therefore, $\text{list}(b') \neq \phi$, and from Claim (i) follows that $\text{list}(b')$ contains the route-designator $\{R'(b'), b', R\}$. From Claim (iii) follows that $\text{list}(s)$ contains this route-designator as well, and therefore $\mathcal{E}$ has the edge $R'(b') \xrightarrow{b'} R$, in contradiction to our assumption.

Let $R \in \mathcal{V}$ be a ring with more than one incoming edge. Let two of these edges be $R_1 \xrightarrow{b_1} R$ and $R_2 \xrightarrow{b_2} R$. This implies that $\text{list}(s)$ contains the route-designators $\{R_1, b_1, R\}$ and $\{R_2, b_2, R\}$. From Claim (ii) follows that $R'(b_1) = R_1$ and $R'(b_2) = R_2$. Since a bridge has only one primary ring, $b_1 \neq b_2$ holds. Claim (ii) also implies that both $b_1$ and $b_2$ have sent a \textsc{search} on $R$, contradicting the fact that exactly one \textsc{search} is sent on each ring (Lemma 2 in Part I).

(3) First note that (1) above implies that node $R''(s)$ cannot be a part of a directed cycle in $(\mathcal{V}, \mathcal{E})$.

Now, suppose that the graph has the following directed cycle $R_1 \xrightarrow{b_1} R_2 \xrightarrow{b_2} \cdots \xrightarrow{b_N} R_1$, where $N \geq 2$ and for every $1 \leq i \leq N$, $R_i \in \mathcal{R}$ holds. This implies that $\text{list}(s)$ contains the route-designators $\{R_1, b_1, R_2\}$, $\{R_2, b_2, R_3\}$, \ldots, $\{R_N, b_N, R_1\}$. From Claim (ii) follows that for every $1 \leq i \leq N$, bridge $b_i$ has sent a \textsc{search} on ring $R_{(i \mod N) + 1}$. Without loss of generality, let $b_1$ be the first bridge from the group $b_1, \ldots, b_N$ that has sent a \textsc{search} on its $R''$. Since $b_1 \neq s$, bridge $b_1$ has received a \textsc{search} on $R'(b_1)$ beforehand. However, this \textsc{search} was sent by $b_N$, contradicting the fact that $b_1$ is the first bridge from the group to have sent a \textsc{search}.

(c) Each leaf $R$ in the \textit{routing-tree} $(\mathcal{V}, \mathcal{E})$ contains a destination from $G$. 

failures are as follows:

It is important to stress that only the instances of the protocol that encounter errors or failures other than some cases of transmission errors and bridge failures may lead the protocol into deadlock conditions.

3.1 Possible Hazards Caused by Errors and Failures

The present section introduces the recovery mechanism for the ERD-protocol. As will be shown, some cases of transmission errors and bridge failures may lead the protocol into deadlock conditions. It is important to stress that only the instances of the protocol that encounter errors or failures are affected. All other instances propagate independently, without being disturbed. The recovery mechanism is designed to detect errors and failures that affect some instances of the ERD-protocol, and to allow recovery for the affected instances.

3 The Recovery Mechanism

Theorem 1 above addresses the case where list(s) ≠ φ upon protocol termination. In those cases where list(s) = φ holds, Claim (iii) in Theorem 1 implies that no member of G appears in rings other than ringb(s). Therefore, in such a case, station s should act as follows:

- If the search sent by s on R''(s) has been received back with r₁ = 1, then members of G do exist on R''(s). Therefore, a data-frame from s to G should be sent by s on its local ring with an empty routing-information field.

- If the search sent by s on R''(s) has been received back with r₁ = 0, then no member of G exists in the multiple-ring network. Therefore, no data-frame from s to G should be sent.

Proof: Let R be a leaf in the routing-tree that contains no destination in G. Since R has an incoming edge in the routing-tree, say R₁ → R, list(s) contains the route-designator \{R₁, b, R\}. This route-designator can be created only by bridge b, upon receiving a response with list ≠ φ in states 5 and 6 (see Table 1(b)). This is because the only other way is for b to receive back the search it sends on R with r₂ = 1, contradicting our assumption that R contains no destination in G. However, from Claim (i) above follows that the first three identities in the list field of the response received by b form a route-designator whose left ring is R. This implies that R has an outgoing edge in G, namely it is not a leaf, contradicting our assumption.

Theorem 1 above addresses the case where list(s) ≠ φ upon protocol termination. In those cases where list(s) = φ holds, Claim (iii) in Theorem 1 implies that no member of G appears in rings other than ringb(s). Therefore, in such a case, station s should act as follows:

- If the search sent by s on R''(s) has been received back with r₁ = 1, then members of G do exist on R''(s). Therefore, a data-frame from s to G should be sent by s on its local ring with an empty routing-information field.

- If the search sent by s on R''(s) has been received back with r₁ = 0, then no member of G exists in the multiple-ring network. Therefore, no data-frame from s to G should be sent.

3.1 Possible Hazards Caused by Errors and Failures

The possible hazards to the new route discovery protocol caused by transmission errors and bridge failures are as follows:

- bridge failure: A bridge b that fails before sending a response frame on its primary ring R'(b) leads the protocol instance into a deadlock condition. This is because the source s, as well as all bridges on the path from b to s on the feedback-tree T induced by the protocol cannot terminate the protocol before b sends a response(0) on R'(b). For example, consider the feedback-tree of Fig. 3. Here, bridge b₃ cannot terminate the protocol before receiving the response(0) sent by b₃ on this ring. If bridge b₃ disconnects from the network before sending such a response, then b₃ stays in state 7 forever. This implies that b₃ does not send a response(0) on its primary ring R₂ and therefore bridge b₂ stays in state 6 forever. Similarly, bridge b₄ waits for a response(0)
on its primary ring forever (deadlock in state 7) and the source \( s \) waits for \( \text{RESPONSE}(0) \) on its secondary ring forever (deadlock in state 6). Recall that \textit{deadlock} in this context means that bridges that are alive (or the source) cannot terminate a \textit{given instance} of the protocol; it can certainly proceed with other instances of the protocol.

- \textit{a transmission error}: Suppose that bridge \( b \) sends \( \text{RESPONSE}(0) \) on \( R'(b) \), and that a transmission error changes this frame before some other bridge repeats this \( \text{RESPONSE} \) as \( \text{RESPONSE}(1) \). The results of such a case are exactly the same as in (a) above, namely a deadlock. One may consider a solution whereby when bridge \( b \) receives its \( \text{RESPONSE} \) back and finds out that this frame has been corrupted, \( b \) sends a new \( \text{RESPONSE}(0) \). However, bridge \( b \) may not know whether the error has occurred \textit{before} or \textit{after} some other bridge had repeated it as \( \text{RESPONSE}(1) \). In the first case a retransmission of \( \text{RESPONSE}(0) \) is necessary, while in the second such a retransmission is wrong and may distort the results of the protocol.

In order to prevent deadlock conditions, the protocol should be extended to allow network stations to detect all possible hazards and to initiate a recovery protocol.

### 3.2 Error and Failure Detection

The \textit{presents} section introduces the method for error and failure detection. The next section presents the recovery protocol, to be initiated upon error or failure detection.

Error detection is very simple: a bridge or the source \( s \) that sends a \( \text{SEARCH} \) or a \( \text{RESPONSE} \) frame can check the \textit{Frame Check Sequence (FCS)} field of its frame upon receiving it back, and find out whether the frame has been disrupted. A station that finds that its frame has changed due to some error, initiates the recovery protocol.

The above solution assumes that the response bits, \( r_1 \) and \( r_2 \) in \( \text{SEARCH} \) frames and \( r_1 \) in \( \text{RESPONSE} \) frames, are also covered by the \textit{FCS}. This is because an unrecognized error in \( r_1 \) leads the protocol into deadlock and an unrecognized error in \( r_2 \) distorts the results of the protocol. For example, if some bridge \( b \) sends on \( R'(b) \) a \( \text{RESPONSE} \) with \( r_1 = 0 \) and a transmission error changes \( r_1 \) to 1, a deadlock arises. However, covering the response bits by the \textit{FCS} is not a trivial task since this would require any bridge that changes \( r_1 \), as well as a non-bridge station that changes \( r_2 \), to recalculate and change the \textit{FCS}. An alternative strategy is to keep \( r_1 \) and \( r_2 \) outside the range of the \textit{FCS}, but to represent these fields with \( n > 1 \) bits rather than one bit (as [5] suggests in another context). From the \( 2^n \) possible values of each field, \( n \) consecutive 0's represents 0 while \( n \)
consecutive 1's represents 1. The remaining $2^n - 2$ vectors occur only as a result of a transmission error. For $n$ sufficiently large, a sequence of errors that changes $r_1 = 0$ to $r_1 = 1$ or vice versa is unlikely.

A failure detection is more complicated. For this purpose, the bridges and the source maintain a timer, called Protocol Termination Time (PTT), associated with each instance of the ERD-protocol. A source $s$ that initiates an instance of the ERD-protocol expects it to terminate at most $\tau_p + \tau_f$, seconds after receiving back the first SEARCH frame, where $\tau_p$ and $\tau_f$ are the upper bounds for the time interval of the propagation and the feedback phases respectively. These values are determined according to the configuration of the multiple-ring network, as explained in Section 3.4. When the source receives back the SEARCH that initiates the ERD-protocol, it sets PTT to $\tau_p + \tau_f$. If the source does not terminate the protocol before PTT expires, it initiates the recovery protocol.

In some rare situations, it might happen that the source is not available when a recovery is required. Such a situation is very uncommon, since it can arise only if at least two failures take place during the execution of the same ERD-protocol instance. This is the situation, for example, when a bridge failure leads the ERD-protocol instance into a deadlock and the source fails too. In order to prevent a deadlock in these rare cases, every bridge that enters a route discovery protocol sets its PTT to $\tau_p + 2 \cdot \tau_f$. This is the latest time when the bridge expects to exit the protocol, due either normal propagation ($\tau_p + \tau_f$) or to the recovery protocol (an additional $\tau_p$, since the recovery protocol can be initiated in any point of the ERD-protocol and its propagation in the entire network is as rapid as the propagation of the ERD-protocol). When PTT expires, the bridge initializes the recovery protocol.

### 3.3 The Recovery Protocol

The recovery protocol informs all bridges in the multiple-ring network about a failure of an instance of the ERD-protocol. The recovery protocol ensures that after its execution, all bridges and the source are in state 0 (idle) for that instance of the ERD-protocol, and no frame of that instance exists in the network.

The recovery protocol is exactly the discovery protocol suggested in [4]. As discussed in Part I [1], this protocol has some deficiencies, most notably its large communication cost. However, this deficiency is less important when the protocol is used for recovery purposes, because the recovery protocol is invoked very infrequently. For example, consider a multiple-ring network with 50 fiber-optic rings whose bit-error rate is $10^{-14}$. Suppose also that there are 150 bridges in the entire network, 100 stations in each ring and that the average length of the SEARCH and RESPONSE frames is 1000 bits. In such a case, the probability of an error during a given instance of the ERD-protocol is less than $2 \cdot 10^{-7}$. The probability that some bridge would fail during such an instance is much smaller, even if the multiple-ring network contains many bridges.

The recovery protocol uses RECOVERY(list) frames, where list is a variable length field containing a list of rings. In addition, a RECOVERY frame contains the identities of the source and the destination of the ERD-protocol instance it belongs to. The recovery protocol proceeds as follows (see Fig. 4). For a given ERD-protocol instance, whenever a bridge or the source detects a corrupted
SEARCH or RESPONSE frame that had been sent by itself, or when its PTT expires, it enters a state, called TRIGGER-RECOVERY (state 9 of the protocol), where it intends to start a recovery protocol for that instance. In this state it waits for access to each ring it is connected to, in order to send a RECOVERY frame on that ring. If the bridge or the source in TRIGGER-RECOVERY state receives in the interim a RECOVERY frame sent by another node for the same ERD-protocol instance, it leaves the TRIGGER-RECOVERY state (state 9) and returns to IDLE state of that instance (state 0). Otherwise, when it gets access to an attached ring, it sends a RESPONSE(list) frame on that ring, where list contains the identities of its attached rings, namely R' and R" if it is a bridge and R" for the source. After sending a RECOVERY frame on each attached ring, the recovery initiator moves to IDLE mode. If a source or a bridge is in a given instance of the ERD-protocol (states 1-8) and receives a RECOVERY frame for that instance, it returns to the IDLE state (state 0). Fig. 4 shows the ERD-protocol together with the recovery protocol at a bridge. The situation for the source is very similar: the only difference is that the source moves from IDLE (state 0) to the protocol upon sending (rather than receiving) a SEARCH, and that it moves from the protocol to IDLE upon receiving RESPONSE(0) (rather than sending such a frame).

As shown above, a bridge or the source that receives a RECOVERY frame in states 1-8 or in the TRIGGER-RECOVERY state, enters state 0 (IDLE). The receipt of a RECOVERY frame in state 0 results in no transition. In addition to the possible transition it makes, a bridge is required to process every received RECOVERY frame, independently of its state. The processing of a RECOVERY(list) frame by bridge b is as follows:

for every ring R connected to b do

if R ∉ list, send RECOVERY(list|R) on R.
In order to ensure the robustness of the recovery protocol, a bridge that receives its recovery back uses the Frame Check Sequence field to detect an error, and the frame is retransmitted if an error is detected.

The source is allowed to start a new ERD-protocol for the considered instance \( \tau_p \) or more seconds after having received the first recovery for that instance. This ensures that the new ERD-protocol finds all bridges in idle state (state 0) for that instance.

As explained in Section 3.1, in those rare cases when the source is not available to initiate a recovery, the timer PTT may expire at several bridges, and those bridges initiate the recovery protocol. Since different instances of the recovery protocol for the same ERD-protocol instance do not merge, each protocol propagates independently of the others, and the recovery penalty increases. Although such a situation is very uncommon, since it requires at least two stations to fail during the protocol execution, it is desirable to restrict the number of independent executions of the recovery protocol as much as possible. This is done in the following way. A bridge whose PTT has expired, and therefore enters trigger-recovery state, but receives a recovery frame before sending one of its own, returns to idle state without initializing a recovery. Since normally PTT expires at different times in different bridges, this strategy decreases the number of independent instances of the recovery protocols for the same ERD-protocol instance.

### 3.4 Protocol Termination Time

As explained before, the failure detection mechanism requires knowledge of an upper bound for the protocol termination time. The purpose of the present section is to find such an upper bound.

Recall that the protocol consists of two logical phases. In the propagation phase, search frames, that traverse each ring exactly once, build the feedback tree. In the feedback phase, response frames are sent from the leaves to the root of the feedback tree.

Let \( \tau_b \) denote the maximal interval that elapses from the time when a frame is placed in a bridge transmission queue until it is received back after transmission and one round trip. This time includes the queueing delay at the bridge, the transmission period and the propagation delay. The maximal queuing delay can be extracted from the transmission rate, the bridge capacity and the maximal length of a bridge queue.

We first determine an upper bound for the propagation phase, \( \tau_p \). The propagation phase starts when the source receives back the first search frame, and completes when every ring in the multiple-ring network had been traversed by a search. Note that the starting point of \( \tau_p \) is taken as the time when the source receives its search frame back, and not when it enqueues this frame for transmission. This saves the need for an additional parameter, \( \tau_b^* \), the maximal interval that elapses from the time when a frame is placed in a non-bridge transmission queue until it is received back after transmission and one round trip. Note that normally \( \tau_b \ll \tau_b^* \), since the access priority is higher for bridges than for non-bridges.

**Claim 1:** Denote by \( D \) the diameter of the multiple-ring network. Then \( \tau_p \leq (D-1) \cdot \tau_b \) holds.

**Proof:** Consider some ring \( R \) in the multiple-ring network. We need to show that at most \( (D-1) \cdot \tau_b \)
seconds after the source receives back the first \textsc{search} frame, such a frame had traversed ring $R$. Since the network diameter is $D$, there exists a route between the source ring $R_s$ and $R_i$ that consists of $D$ or less rings, including $R$ and $R_s$. Let $R_1 \triangleleft R_2 \triangleleft \ldots \triangleleft R_{i-1} \triangleleft R_i \triangleleft \ldots \triangleleft R_N$, where $R_1 \equiv R_s$, $R_N \equiv R$ and $N \leq D$, be such a route. Since we define the starting point of the propagation phase as the time when the source $s$ receives back its \textsc{search} frame on $R_s$, then already before the propagation phase starts, bridge $b_1$ receives the \textsc{search} sent by the source, and puts such a \textsc{search} in the queue of its other ring $R_2$. By induction, bridge $b_i$ receives the first \textsc{search} frame no later than $(i-1)\tau_b$ seconds after the propagation phase starts. This is because by the induction assumption at most $(i-2)\tau_b$ seconds after the propagation phase starts, bridge $b_{i-1}$ receives the first \textsc{search} frame. Note that bridge $b_{i-1}$ may receive the first \textsc{search} either from ring $R_{i-1}$ or from ring $R_i$. If $b_{i-1}$ receives the first \textsc{search} from ring $R_i$, then bridge $b_i$ receives the first \textsc{search} from the same ring at the same time. However, if bridge $b_{i-1}$ receives the first \textsc{search} from ring $R_{i-1}$, then at most $\tau_b$ seconds afterwards a \textsc{search} will be sent on $R_i$, either by $b_{i-1}$ or by some other bridge.

This implies that bridge $b_{N-1}$ receives the first \textsc{search} no later than $(N-2)\tau_b$ seconds after the propagation phase starts. If this \textsc{search} was not received from ring $R_N$, then at most $\tau_b$ seconds later, a \textsc{search} will be sent on $R_N$, either by $b_{N-1}$, or by some other bridge. Thus, a \textsc{search} frame is sent on ring $R$ no later than $(N-1) \cdot \tau_b$ seconds after the propagation phase starts, and since $N \leq D$ holds, the proof is completed.

We next determine an upper bound for the feedback phase, $\tau_f$, the time interval between the termination of the propagation phase and the termination of the protocol. Note that \textsc{response} frames may be transmitted in the protocol before the propagation phase ends, but with the above definition $\tau_p + \tau_f$ is an upper bound to the total protocol time.

Let $L$ denote the number of edges in the longest directed path from a leaf to the source $s$ in the feedback-tree. We now divide the nodes in the feedback-tree into levels, according to their distance from the source $s$. Level $L$ consists of the nodes that have a directed edge to the source $s$ in the feedback-tree, and level $i$, for $1 \leq i < L$, consists of all the nodes that have a directed edge to a node in level $i+1$. Recall that each node has only one outgoing edge in the feedback-tree, and therefore it belongs to exactly one level. For example, in the feedback-tree of Fig. 3, $L = 4$, $b_4$ is in level 4, $b_2$ is in level 3, $b_1$ and $b_3$ belong to level 2 and $b_5$ belongs to level 1.

\textbf{Claim 2:} A bridge $b$ in level $i$ sends a \textsc{response}(0) on its primary ring, receives it back and terminates the protocol at most $(i \cdot \tau_b)$ seconds after the propagation phase ends.

\textit{Proof:} By induction on $i$. At most $\tau_b$ seconds after the propagation phase ends and all bridges recognize a \textsc{search} on their secondary ring, every bridge in level 1 sends a \textsc{response}(0) on its primary ring and receives it back, since a bridge in this level has to wait for no \textsc{response}(0) before sending a \textsc{response} and terminating the protocol. Recall that in some feedback-trees it may happen that bridges send a \textsc{response}(0) before the propagation phase ends, but this does not affect the result. Assume the claim holds for all bridges in level $i-1$. This implies that no later than $\tau_b \cdot (i-1)$ seconds after the propagation phase ends, a bridge $b$ in level $i$ receives a \textsc{response}(0) sent by the

16
bridges in level \( i - 1 \) that have a directed edge to \( b \) in the feedback-tree. When this happens, bridge \( b \) puts a RESPONSE(0) frame in the queue of its primary ring, and in the ensuing \( \tau_b \) seconds this frame is transmitted and received back by \( b \).

The conclusion of Claim 1 and Claim 2 is that \( \tau_p \leq (D \cdot \tau_b) \) and \( \tau_f \leq (L \cdot \tau_b) \). The values of \( L \) and \( D \) may vary, but they are always bounded respectively by \( B_{\text{max}} \), the maximal number of bridges in the network, and \( R_{\text{max}} \), the maximal number of rings in the network. Thus \( \tau_p \leq \tau_b \cdot |R_{\text{max}}| \) and \( \tau_f \leq \tau_b \cdot |B_{\text{max}}| \).

Note that \( |B| \geq |R| - 1 \) always holds. Fig. 5(a) shows a case where \( |B| \approx |R| \), whereas an opposite case, with \( |B| \gg |R| \), is shown in Fig. 5(b). When \( |B| \gg |R| \), the contribution of \( \tau_f \) to the protocol termination time is large, namely the feedback phase might be much slower than the propagation phase. For example, in the multiple-ring network of Fig. 5(b), \( \tau_p = 1 \cdot \tau_b \), whereas \( \tau_f = N \cdot \tau_b \).

It turns out that by making a small modification in the bridge algorithm, thus creating a new version of the ERD-protocol, referred to as early termination version, it is possible to bound \( \tau_f \) by \( \tau_b \cdot |R_{\text{max}}| \) rather than by \( \tau_b \cdot |B_{\text{max}}| \) in order to speed up the termination of the protocol. Consider again the feedback-tree in Fig. 3. The path \( b_1 \to b_2 \to b_4 \to s \) indicates that in ring \( R''(s) \), bridge \( b_2 \) must wait for the RESPONSE(0) of \( b_1 \) before sending RESPONSE(0) and terminating the protocol. Similarly, bridge \( b_4 \) must wait for the RESPONSE(0) of bridge \( b_2 \) and \( s \) must wait for the RESPONSE(0) of bridge \( b_4 \). This sequential termination in every ring is the reason for the possible long period of the feedback phase. Recalling from Part I [1] Section 2 conditions \( A \) and \( B \) for termination,
the idea behind speeding up termination is that a bridge can in fact terminate the protocol even if condition A is not satisfied, namely while it is still waiting for a RESPONSE(0) on its primary ring. The change is that a bridge that terminates the protocol when only condition B is satisfied (state 7 in Table 1(a)), should send on its primary ring RESPONSE(1) instead of RESPONSE(0).

For example, in the abovementioned situation, bridge b₄ sends RESPONSE(1) before receiving the RESPONSE(0) of b₂. The source s receives this RESPONSE, and since τ₁ ≠ 0 holds, s knows that the protocol is still executed. Later, bridge b₄ sends a RESPONSE(0), and bridge b₂ repeats this as RESPONSE(1) and notes that condition A is satisfied. Afterwards, when b₂ receives a RESPONSE(0) on its secondary ring too, it sends on $R''(s)$ a RESPONSE(0) and terminates the protocol. Since bridge b₄ has terminated the protocol, it repeats this RESPONSE(0) unchanged, thus the source s receives it, and knows that the protocol is completed. Note that if bridge b₂ receives a RESPONSE(0) on its secondary ring before receiving the RESPONSE(0) sent by b₁ on its primary ring, it sends on the primary ring a RESPONSE(1) rather than RESPONSE(0) and terminates the protocol. In such a case, the RESPONSE(0) sent by b₁ is repeated by b₂ and b₄ unchanged, the source s receives it as RESPONSE(0) and terminates the protocol. In any case, at most $\tau_s$ seconds after the propagation phase terminates, bridges b₅, b₁ and b₄ send a RESPONSE and terminates the protocol, and at most $2 \cdot \tau_s$ seconds after the termination of the propagation phase, bridge b₂ sends a RESPONSE and terminates the protocol.

An interesting property of this version is that a bridge b for which list(b) = nil, and for which condition A is not satisfied, may terminate the protocol without sending a RESPONSE on its primary ring. In such a case, the bridge just exits the protocol and enters state 0. For example, this is the case of bridge b₄ in the above scenario. The only difference between this early termination version and the basic ERD-protocol is in state 7 of Table 1(b). A bridge b that enters state 7 while list(b) = nil, makes a transition to state 0; otherwise, it acts according to the following (changes compared to state 7 in Table 1(a) are boldface):

<table>
<thead>
<tr>
<th>State</th>
<th>State Description</th>
<th>Event</th>
<th>Reaction</th>
<th>Next</th>
</tr>
</thead>
<tbody>
<tr>
<td>7</td>
<td>- waiting for RESPONSE(0) on $R'$ (condition A)</td>
<td>receive RESPONSE(0, list) on $R'$</td>
<td>repeat RESPONSE(1, list) on $R'$</td>
<td>8(a)</td>
</tr>
<tr>
<td></td>
<td>- waiting for nothing in $R''$</td>
<td>access to $R''$</td>
<td>send RESPONSE(1) on $R'$</td>
<td>8(b)</td>
</tr>
</tbody>
</table>

In order to see that in the early termination version $\tau_f \leq |B_{\text{max}}| \cdot \tau_h$, we redefine L as the maximal number of solid edges in a directed path in the feedback-tree. We consider again a division of the network bridges into levels. Level $L^*$ contains the bridge that has a directed solid edge to the source s. Thus, $L^* \leq |R_{\text{max}}|$ must hold. A bridge b is in level $i \leq L^*$ if it has a dashed edge to another bridge in level $L^*$, or if it has a solid edge to a bridge in level $i + 1$. For example, consider again the feedback-tree of Fig. 3. Here $L^* = 2$, bridge b₄ is in level 2, and therefore, bridges b₂ and b₁ are also in this level. Since b₂ is in level 2, bridge b₃, that has a directed edge to b₂, is in level 1, and bridge b₅, that has a directed dashed path to b₃ is in level 1 as well.

**Claim 2**: Consider the early termination version of the ERD-protocol. A bridge b in level i sends
a \texttt{RESPONSE(0)} on its primary ring, receives it back and terminates the protocol at most \((i \cdot \tau_b)\) seconds after the propagation phase ends.

\textbf{Note:} This claim has the same statement as in Claim 2. The difference is induced from the fact that in Claim 2, the number of levels \(L\) is bounded by the maximal number of edges, either \textit{solid} or \textit{dashed}, in the feedback-tree, whereas here \(L^*\) is bounded by the maximal number of \textit{solid} edges in the feedback-tree. This implies that \(L \leq |B_{\text{max}}|\), while \(L^* \leq |R_{\text{max}}|\).

In order to prove Claim 2*, Lemmas 3, 4 and 5 of Part I should be restated and proved for the new version. To stay within the paper length restrictions, we omit this proof.

The conclusion of this section is that (1) \(\tau_p \leq |R_{\text{max}}| \cdot \tau_b\); (2) \(\tau_f \leq |R_{\text{max}}| \cdot \tau_b\) or \(\tau_f \leq |B_{\text{max}}| \cdot \tau_b\), depending whether the early termination version is considered or not, respectively.

4 \textbf{High-Speed Processing}

The \textit{ERD-protocol} requires more processing by the bridges than the protocol in [4]. Moreover, most of the processing must be done while forwarding the frame from the input to the output channel, and thus it must be performed very fast. These time constraints may make implementation of the protocol difficult in networks with very high speed rings.

As explained in Part I [1], it is possible to add some artificial delays by placing several unused bits before the response field \(\tau_f\) in the \texttt{SEARCH} and \texttt{RESPONSE frames}. In this section we introduce another version of the protocol, to be used in the case when the number of required unused bits is too high. This version enables the bridges to repeat every \texttt{SEARCH} or \texttt{RESPONSE} frame with no on-line processing, except for frame recognition.

As shown later, in the high-speed processing version \(\tau_f \leq |B_{\text{max}}| \cdot \tau_p\) holds, whether the early termination version is used or not. On the other hand, high-speed processing combined with the early termination version results in higher communication cost. Therefore, we shall restrict high-speed processing to the case when early termination is not used.

The idea is that instead of making an on-line decision, while a \texttt{SEARCH} or a \texttt{RESPONSE} frame traverses the bridge, whether this frame should be repeated with \(\tau_1 = 1\) or \(\tau_1 = 0\), the bridge repeats every frame with \(\tau_1 = 1\). In addition, the bridge copies the frame for later processing. Afterwards, when it has sufficient time, the bridge checks to which protocol does the frame belong to and what is its state in this protocol. If the outcome of this check requires the frame to have been forwarded with \(\tau_1 = 1\), no further action is necessary except for performing the necessary state transition. On the other hand, if the frame had to be forwarded with \(\tau_1 = 0\), the bridge waits to next get access to the same ring and sends a \texttt{RESPONSE(0)} frame with \texttt{list} = \emptyset. Note that a \texttt{RESPONSE(0)} frame is sent in both cases, whether the mistakenly sent frame was a \texttt{SEARCH(0, r_2)} or a \texttt{RESPONSE(0)}.

It turns out that every bridge may make such a 'mistake' only once for each instance of the \textit{ERD-protocol}. Moreover, this 'mistake' may take place only in the secondary ring of a bridge. Thus, this new version requires every bridge to send at most one \texttt{RESPONSE frame} on its secondary ring in addition to the \texttt{RESPONSE} frame it sends on its primary ring. Hence, at most \(2 \cdot |B| + |R|\)
frames are sent by each instance of the ERD-protocol.

The following example demonstrates the execution of the high-speed processing version, and shows the communication cost. Consider the ring $R$ in Fig. 6 and suppose that this ring is a part of a multiple-ring network. Consider a certain instance of the ERD-protocol, processed by the network stations according to the high-speed processing version. Let $b_1$ be the bridge that sends the SEARCH on $R$. Suppose that for bridge bridge $b_4$ this is the first received SEARCH, whereas bridges $b_2$, $b_3$ and $b_5$ have already received a SEARCH on their other ring before.

According to the high-speed processing version, the SEARCH$(0, r_2)$ sent by $b_1$ is repeated by $b_2$ as SEARCH$(1, r_2)$. All other bridges repeat this SEARCH unchanged, so $b_1$ receives back a SEARCH with $r_1 = 1$. Following the transmission of this SEARCH, bridge $b_4$ enters the protocol. Since it has received the SEARCH with $r_1 = 1$, it waits for a RESPONSE$(0)$ on $R = R'(b_4)$ before terminating the protocol. Bridge $b_1$, the SEARCH sender, must also wait for a RESPONSE$(0)$ on $R = R''(b_1)$ before it terminates the protocol. When bridge $b_2$ realizes that the received SEARCH should have been repeated unchanged, namely with $r_1 = 0$, it corrects the ‘mistake’ by sending a RESPONSE$(0)$ on $R$. Bridge $b_3$ repeats this RESPONSE as RESPONSE$(1)$, but later, when it checks its condition in the associated instance, and finds out that $R$ is its secondary ring for that instance, it corrects the ‘mistake’ by sending a RESPONSE$(0)$. Bridge $b_4$ repeats this RESPONSE$(0)$ as RESPONSE$(1)$, and notes that condition A is satisfied. If bridge $b_4$ does not wait for a RESPONSE$(0)$ on $R''(b_4)$, then it sends a RESPONSE$(0)$ on $R$ and terminates its part in the protocol. If it waits for a RESPONSE$(0)$ on $R''(b_4)$, it will send a RESPONSE$(0)$ on $R$ after receiving a RESPONSE$(0)$ on $R''(b_4)$.

According to the original version, the RESPONSE$(0)$ sent by $b_4$ on $R$, would have been repeated unchanged by bridge $b_5$. However, according to the high-speed processing version, bridge $b_5$ repeats it as RESPONSE$(1)$, and later corrects itself by sending a RESPONSE$(0)$ of its own. The latter is received by bridge $b_1$, thus completing the requirements for condition B.

The last example shows that a bridge makes no ‘mistakes’ on its primary ring (bridge $b_4$ in our example). It also shows that a bridge that does not send a SEARCH on its secondary ring, makes exactly one ‘mistake’ on that ring: bridge $b_2$ has repeated the received SEARCH with $r_1 = 1$ rather than with $r_1 = 0$; bridge $b_3$ has repeated the RESPONSE$(0)$ sent by $b_2$ as RESPONSE$(1)$ rather than RESPONSE$(0)$; bridge $b_5$ has repeated the RESPONSE$(0)$ sent by $b_4$ as RESPONSE$(1)$ rather than RESPONSE$(0)$.
RESPONSE(0). On the other hand, a bridge that sends a SEARCH on its secondary ring (like bridge \( b_1 \) in our example) makes no 'mistakes' on that ring.

The last example can also explain why it is not possible to achieve high-speed processing with early termination. If the early termination version is applied, then bridge \( b_4 \), that enters the protocol due to the receipt of the SEARCH(1, \( r_2 \)) sent by \( b_1 \), may send a RESPONSE(1) before receiving RESPONSE(0), and then it terminates the protocol. Afterwards, when bridge \( b_4 \) receives the RESPONSE(0) sent by \( b_3 \) later, it repeats this RESPONSE(0) as RESPONSE(1), as the high-speed processing rules dictate. Thus, it must send another RESPONSE, with \( r_1 = 1 \), in order to enable protocol termination. This means that bridge \( b_4 \) sends an extra RESPONSE frame on its primary ring, but the protocol termination speed is not affected: the SEARCH sender, \( b_1 \), may not terminate the protocol before all bridges on its secondary ring send a RESPONSE(0) frame one after one.

5 Conclusion

The paper has presented some extensions of the ERD-protocol. Section 2 has shown how the ERD-protocol can be used for discovering of a routing-tree from a source to a group of destinations. Since the route discovery version of the ERD-protocol is a special case of the tree discovery version, only the tree discovery version is actually required, and this algorithm should be executed whether the destination is a single station or a group of stations. Therefore, the output of Section 2 is a single protocol that can find a route-description or a tree-description for routing to a single destination or a group of destinations respectively.

The ERD-protocol that can find a route-description or a tree-description was considered as the input to Section 3. The output of this section is two robust protocols: the first protocol, say \( P_1 \), is a robust ERD-protocol that can be used for finding route-description and tree-description; the second protocol, \( P_2 \), is the same as \( P_1 \), but it also has the early termination property.

Section 4 takes as input the \( P_1 \) protocol, and changes it into a third protocol, \( P_3 \), to be used when the multiple-ring network bridges cannot change the bit \( r_1 \) on-line.

The result of Part I and Part II is two protocols:

\( P_2 \): a robust protocol that can find a tree-description or a route-description, has the early termination property (i.e. \( \tau_f \leq |R_{\text{max}}| \cdot \tau_b \)), but requires the bridges to make a local search while repeating a SEARCH or a RESPONSE frame.

\( P_3 \): a robust protocol that can find a tree-description or a route-description, does not have the early termination property (i.e. \( \tau_f \leq |B_{\text{max}}| \cdot \tau_b \)), but enables a bridge that receives a SEARCH or a RESPONSE frame to make its local calculations off-line, after repeating the frame.
References


