Distributed Synchronization Protocols for SDH Networks

Yehuda Afek* Tal Lev†
Computer Science Department
Tel-Aviv University
Israel 69978

Abstract

In this paper we design distributed protocols for synchronization in SDH networks. We start by suggesting a new abstraction to model the communication of control information in SDH/SONET frame headers. The new model is then used in the construction of realistic and fault-tolerant distributed protocols for the timing distribution problem. In this problem the nodes of an SDH/SONET network should be synchronized to transmit at the same rate; otherwise the data carried by the network may be impaired. The presented protocols ensure that (1) all the nodes derive their transmission clock from one source, (2) that this source is the highest quality clock available in the network and (3) that the path along which the clock at a node is derived is the shortest path from the source to the node. The design of the presented protocol employs modern techniques from the field of distributed computing adapted to the model and tools available in a real SDH/SONET network.

Keywords: Synchronization, SDH, SONET, Transmission rates, clock distribution.

*Contact author; Telephone: +972-3-6408952,Fax: +972-3-6409357, e-mail: afek@math.tau.ac.il
†Telephone: +972-3-6723846, Fax: +972-3-9266200, e-mail: levtal@math.tau.ac.il

1 Introduction

SONET (Synchronous Optical Network) and SDH (Synchronous Digital Hierarchy) networks are recently becoming the major infrastructure for digital transmission backbones. These networks are used for CBR (Constant Bit Rate) wide-area services, i.e., to multiplex and convey continuous, constant bit rate streams over metropolitan, national and international areas. These networks are made of nodes, or Network Elements (NEs), connected via high-speed transmission links, usually fibers. The most common topology for SDH networks is the dual ring topology. Other common topologies are uni-directional ring (single ring), chain, and star. Future SDH topologies include inter-connected rings and general (mesh) topologies.

Each node in an SDH network transmits data over its out-going links at a rate determined by an internally generated clock. This clock may be derived from one of several possible sources (see Figure 1): an internal free-running reference clock, a reference clock external to the node (e.g., an atomic reference clock), and clocks recovered from the rates of the transmission links incoming to the node. At each node an independent decision is made by control software as to which timing source to select. This may create several problems:

- Rate mismatches. Although SONET/SDH include a mechanism to compensate for rate mismatches (the mechanism, called "pointer justifications", ensures that no data is lost if the rate differences are within some bound), if the network is not synchronized, *jitter* is generated in the transmission payloads carried by the network, and thus transmission quality is degraded.

- *Timing loops* may be formed in the network, where a timing loop is a cycle of nodes in the network such that each node derives its transmission rate from the rate of its predecessor in the cycle. Such loops lead to large frequency deviations which may be fatal to data integrity.

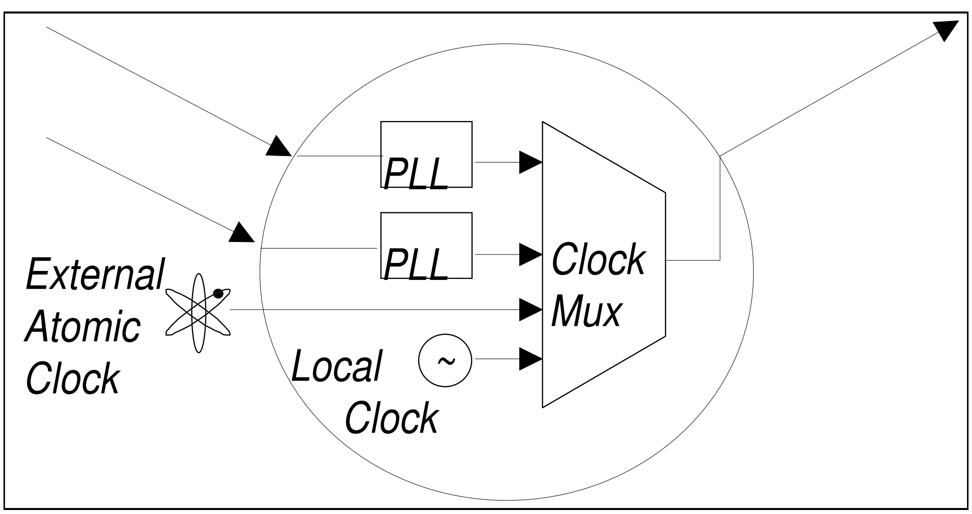

Figure 1: Clock source selection in a node.

It is hence desirable for an SDH network to be *synchronized*, that is, to have all the nodes in the network derive their transmission clocks from one independent clock source.

Since the available clock sources may differ in their qualities (e.g., their absolute frequency accuracy and their long-term stability), it is also desirable to use the highest quality clock source available.

Network synchronization may be achieved by having one node (the *source* node) select its timing source to be this independent, highest-quality clock source, and all the other nodes set their timing sources to be the clocks recovered from the links incoming from the source node (either directly, or in tandem through other nodes). This method creates a *timing distribution tree* in the network.

1.1 Current Status

The standard bodies defining SONET/SDH[1] both specify the synchronization sub-network required to solve the problem, and suggest a mechanism that is supposed to construct such a sub-network. However, the mechanism suggested by the standards is prone to user errors, exhibits unstable behaviors, is fault-intolerant, and requires manual intervention upon any change in the underlying network. In this section we briefly review the specified requirements, and the suggested mechanism.

Standards requirements

The standards specify the following requirements for the synchronization subnetwork:

- The synchronization sub-network should conform to a tree topology: The ETSI standard in [29] states that "synchronization is transmitted through the network via synchronization trails", that is, through the transmission links, and in [30] goes as far as defining that "inter-node distribution conforms to a tree-like topology and enables all the nodes in the SDH network to be synchronized." ANSI, in [31], says that "it is essential to insure that timing loops are not formed."

- This tree should be rooted at a node offering a clock source of the best quality available: ANSI, in [31], says that "the goal of the synchronization reference selection algorithm is to select the best possible timing reference for all [the nodes] in the network." The ETSI standard in [29] requires that "all clocks in the network will be traceable to the network PRC" (Primary Reference Clock).

- The tree should be of a minimal depth: Since "the quality of timing will deteriorate as the number of synchronization links increase", ETSI in [30] requires in addition that "the number of network elements in tandem should be minimized."

[1]Three standard bodies define SDH-like networks: ITU (International Telecommunications Union, formerly known as CCITT - Consultative Committee for International Telephone and Telegraph) and ETSI (European Telecommunications Standards Institute) define SDH, while ANSI (American National Standard Institution) defines the American equivalent of SDH, called SONET (Synchronous Optical NETwork). The relevant standards defining SDH and SONET appear in [25] for ITU, [26] for ETSI and [27] for ANSI. The relevant standards for the *timing function* of SDH and SONET may be found in [28] for ITU, [29, 30] for ETSI and [31] for ANSI.

- Operation under fault conditions: The standards emphasize the importance of proper network synchronization when faults occur. ANSI ([31]) remarks that "although the traffic path will automatically reconfigure (self-heal) under fault conditions, the synchronization reference timing will not automatically reconfigure without appropriate messaging between SONET NEs" and ETSI, in [29], clearly states that "in the event of a failure of synchronization distribution, all network elements will seek to recover timing from the highest hierarchical clock source available."

Standards suggested mechanism

The local mechanism suggested by the standards has two components. First, it is suggested that transmission links carry "an indication of the quality level of the source generating the timing, called Synchronization Status Message (SSM)" ([29]) or "timing quality marker" ([30]), written in the SDH frame's header. Second, an operator-defined "Prioritized List of Synchronization Sources" is set in each node.

The SSM and the list are combined in an algorithm run locally at each node, common to all three standard bodies, that can be summarized as follows: From the candidate sources appearing in a node's list, the node selects the source that offers the highest quality (by the SSM). If more than one candidate source offers this quality, then the node selects the one that is higher up the list.

In order to avoid timing loops, the SSM message can also be assigned the special value "don't use for synchronization." ETSI, in [29], simply suggests that this message "is sent back to the equipment which is the source of the selected timing."

1.2 Drawbacks

The mechanism suggested by the standards suffers from several major drawbacks. First, relying on user-defined priority lists makes this critical mechanism prone to human errors. Even for simple networks the required lists are non trivial, and [31] states that

> ... "careful engineering is required."

Second, each change in the network (e.g., adding a network element or adding an external timing source) may require re-configuration of the lists over the whole network.

Thirdly, the "don't use for synchronization" messaging method suggested by the standards does **not** avoid the generation of timing loops in certain topologies. ETSI itself states that if there is more than one path between two nodes, "the SSM approach cannot, by itself, prevent generation of timing loops" ([29]), and concludes ([30]):

> ... "Where timing quality markers are used in a meshed network, potential problems have been identified ... Network synchronization planners are reminded that careful consideration is needed when using timing quality markers."

We further claim that even if the tables are properly configured, using them during failures may lead not only to sub-optimal synchronization networks, but also to unstable ones. For example, consider the simple ring in Figure 2, which has exactly one external source of quality 1, connected to node 1. The available sources for nodes 2 and 3 are the clockwise and counter-clockwise incoming links (we denote those as CW and CCW, respectively) and their internal oscillators (denoted INT). Considering that the most probable faults are single link faults, the priority list of sources (as defined by the standards) in node 2 should be CW, CCW, INT, and the list in node 3 should be CCW, CW, INT. But if link 1-2 fails concurrently with link 1-3, then node 2 will switch to CCW and node 3 to CW. Trying to avoid a timing loop, both nodes will transmit a "don't use for synchronization" SSM to each other, so both will switch to INT and transmit its quality level to each other. Now both nodes receive from the line a quality level equal to their INT quality; and since the line sources are higher up on their priority list, node 2 will revert to CCW and node 3 to CW. The process of switching back and forth between sources may never terminate, so even in this trivial topology, an unstable system may result.

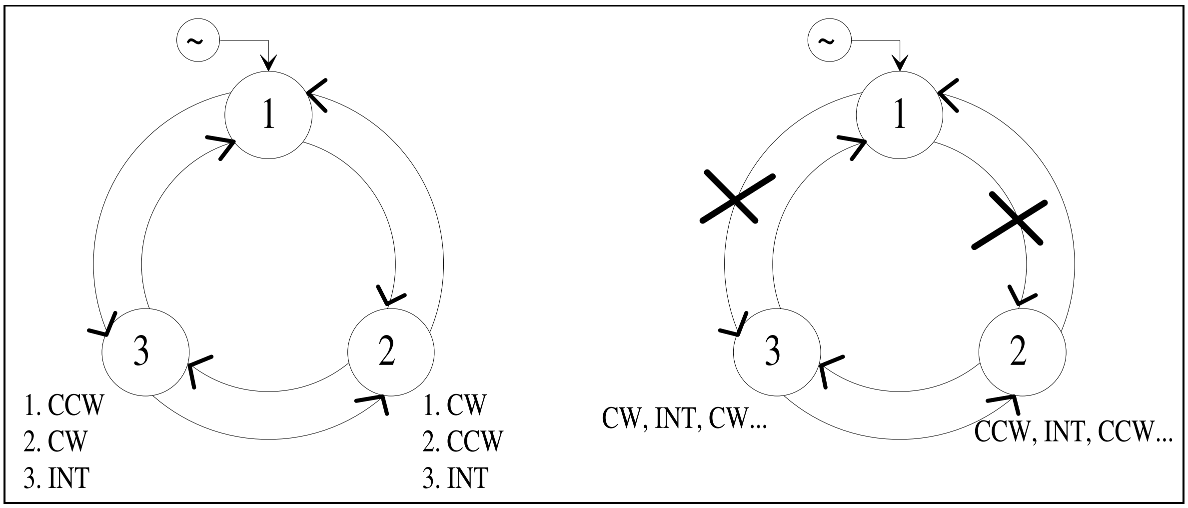

Figure 2: Unstable timing created when faults occur

Neither of the current SDH/SONET standards includes a complete solution for network synchronization, and none of them guarantees proper operation when faults occur. With rings having tens of nodes already deployed, and with the continuous increase in network size, the need for *automatic* management is apparent. Currently, SDH networks are managed by a central management system. However, the need for *distributed* control arises when faults occur in a network (the central management system may be disconnected from parts of the network), and as network sizes increase.

1.3 A Solution

In this paper we suggest a solution to the timing distribution problem in SDH/SONET networks. The solution is built on distributed algorithms, tailored to run in SDH/SONET networks, that construct a spanning tree (or forest) rooted at the node(s) with the best clock source. All the other nodes in the network recover their transmission rates from the data received along their parent links in the tree(s).

For the design of these algorithms for SDH/SONET networks we suggest an abstraction model of these networks. This new model supports the easy implementation of novel protocols, in software, over existing hardware. This enables SDH networks to evolve smoothly, alleviating the need for frequent, expensive hardware modifications.

The new model enables the development of distributed SDH management functions, using techniques of dynamic network protocols and self stabilization that have been developed in the past few years, such as [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14]. The approach of abstracting new technologies in a distributed computing model was previously applied to high speed networks in [15] and [16]. However, the abstraction in [15, 16] applies mainly to the way packets are routed in fast datagram networks, while here we are concerned with control messages over a continuous stream.

After presenting the model abstraction we present in this paper several solutions to the timing distribution problem: First, a complete fault-tolerant and self-stabilizing solution for either bi-directional or uni-directional rings is presented (Section 3). This solution is constructed of layered protocols. The lower-level protocols include some new building blocks, including a self-stabilizing uni-directional ring reset protocol and a self-stabilizing ring integrity detection protocol, both using short, constant-size messages. The upper level is based on a building block that enumerates the nodes of a uni-directional ring using constant-size messages in $\log n$ rounds, i.e., in $O(n \log n)$ messages. Secondly, we employ self-stabilizing techniques to construct algorithms for arbitrary topology bi-directional networks (Section 4). Finally we conclude (Section 5) by a discussion of the necessary procedures for the construction of such algorithms in uni-directional networks. In the appendix, a short description of SDH is given. Proofs of some properties of the presented algorithms are also differed to the Appendix.

2 Model and New Abstraction

Each node in an SDH/SONET network continuously sends a stream of data frames in a fixed rate of 8000 frames per second. Each frame is partitioned into two parts: Payload - where the data is transferred, and header (overhead) in which control signals, alarms, management and monitoring information is placed (see Appendix A).

Due mainly to the inferiority of processor/memory speeds compared to the transmission speeds in SDH, protocols have to deal with not being able to process header information on every frame. This difference in basic underlying infrastructure makes the use of traditional protocols, such as HDLC, inappropriate in high speed networks. These protocols (e.g., HDLC) require 2-way communications, and are not self-stabilizing [10]. Techniques to express and formulate new protocols under this inferior-processor-speed situation are necessary. This section describes such a model, which in essence, associates a processor-memory element with each unit of the frame header of an SDH network, and models the network communication as essentially inter-process communication without "memory access synchronization."

In most SDH implementations, every node maintains a receive buffer for each incoming link into which the header of each frame that arrives over that link is written (see Figure 3). This buffer is thus over-written every time a frame arrives, regardless of whether the previous content of the buffer was read or not. Thus, if a particular information is placed only in one frame header and the reading agent in the receiving end fails to read it before the next frame header overwrote the buffer, then this information is not observed by the receiver. Hence, some synchronization mechanism is necessary to ensure the correct use of this communication channel (the header channel).

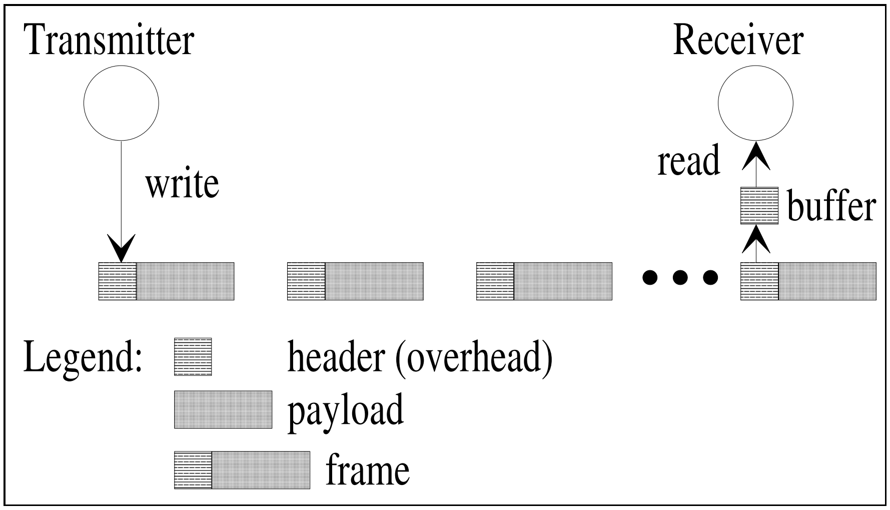

Figure 3: The Model

One approach would be to use a higher level protocol on top of the frame header (such as an alternating bit protocol) to guarantee that each piece of information is received before we send a new one. Alternatively, protocols that use the frame header should be specifically designed to work in that framework. Clearly, using a higher level protocol, such as HDLC, slows down the transmission rate of this channel and introduces extra delays into it.

Here we suggest an abstraction to model the above mechanism in a way that would be convenient for the design of higher level protocols. This model abstracts away the technicalities of the SDH standards and its implementation details thus enabling researchers the ease of developing distributed network management algorithms for these networks.

In our abstraction we consider communication over a fixed number of bits in the frame header as writing and reading to and from a shared register. The transmitting agent writes to the register by setting the corresponding bits in the frame header and the receiving agent reads the register by reading the corresponding bits in the receiving buffer. If the link is operational in both directions then the two agents also share a register in the reverse direction. As before, if the sender writes a new value before the receiver has read the previous one, then the previous value is not observed by the receiver.

2.1 The Timing Distribution Problem

For each node $v \in V$ there exists a list of potential timing sources, L_v, each with a defined level of quality. The maximum list length $|L_v|$ and the number of possible quality levels are constant. Each node, v, in the network transmits data into the links at a rate r_v, derived from one of the timing sources. That is, the transmission rate r_v may be set to any one of the sources listed in L_v. These sources may be:

1. The rate recovered from any one of the node's incoming links.

2. The rate of its internal clock.

3. The rate of an external clock, connected to the node (e.g., an atomic clock).

In order to ensure that a connected set of nodes all transmit at the same rate, we construct a timing distribution tree that spans the set of nodes. The tree is formed by properly setting the timing source in each node. In the tree's root, the protocol sets the timing source to internal (or external, whichever is better). In each of the other nodes, the protocol sets the node's clock to be the clock derived from its tree-parent incoming link.

We consider two variations for the problem:

1. All the nodes of the network are in one such set, so that a timing spanning *tree* with a single root is constructed. This variation conforms to the Synchronous Mode as defined by ETSI in [30], which is "the normal mode of operation within a single operator's domain."

2. The nodes of the network may be partitioned into several such sets, using a timing *forest*, with multiple roots, instead of a tree. This variation conforms to the Pseudo-synchronous Mode as defined by ETSI in [30], which is "the normal mode of operation for the international and inter-operator network."

In the first variation, the protocol should generate a timing spanning *tree* in the network, such that:

- The tree root is one of the nodes that own the best quality source in the network.

- The tree is the most shallow tree among all possible trees rooted at a node with the best-quality source.

In the latter variation, the protocol should generate a *timing forest* in the network, such that:

- The roots are the nodes with the highest quality clock source in the network.

- Each tree is a BFS (Breadth First Search) tree.

- Each node is attached to the tree of the closest root (ties broken arbitrarily).

The protocol should also construct such a tree (or forest) in cases of failures of nodes and links in the network; for the first variation, when, due to link or node failures, the network is partitioned, the protocol should create a *forest* of such trees, so that in each connected component of the network a valid timing tree is created.

During the operation of the protocol, we allow nodes to momentarily derive their clocks in a way that does not conform to the above requirements. This is justified because the hardware timing circuits and generators have very long time constants in their stabilization times, and momentary violations would not adversely affect the operation of the network. Also, the different sources are assumed to be around the same frequency and within a range that would not immediately affect data integrity.

In our protocol for the ring topology, we also assume the following:

- Exactly one of the nodes in the ring is designated as the leader. This requires that the user enters this information as part of the node's installation procedure[2].
- A consistent orientation of the links (clockwise or counter-clockwise) is marked on the link ports at every node. This is ensured from the node's installation procedure and hardware connections.

3 The Protocol for Rings

In this section we propose a timing tree protocol, specifically designed for the ring topology. This proposed protocol does not require that unique IDs are given to nodes, and also operates when failures occur in parts of the ring.

The crux of the ring protocol is a distributed algorithm that enumerates the nodes of a uni-directional ring, using constant-size messages, in accordance with the model (Basic Algorithm A in Section 3.4).

3.1 General Protocol Outline

We use a layered model to describe the protocol. Three layers are defined, with the output of the lower levels being the input to the upper levels. We designate these layers as the *state*, *reset* and *tree* layers (see Figure 4).

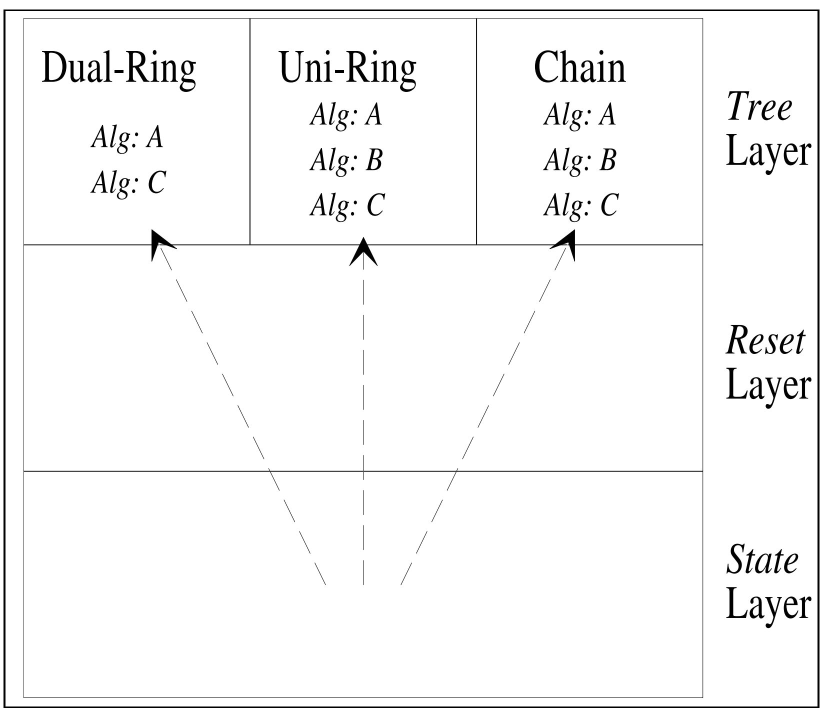

Figure 4: Layers of the Ring Protocol

[2]If multiple leaders are erroneously defined, the network *will* start, and then the higher-level network management system (NM) will be able to detect this error and correct it.

The *state* level is responsible for detecting the state of the ring, i.e., whether a complete ring exists in the clockwise and/or counter-clockwise directions (a complete ring exists in a specific direction iff all the links in that direction are operational). The intermediate *reset* level is responsible for initializing the protocol when configuration changes are detected. The top level constructs the timing distribution tree, and runs one of three different algorithms (according to the state of the ring).

When a complete ring exists, the pre-designated leader coordinates the algorithm: if a ring exists in both direction, it runs the *dual-ring* protocol, and if a ring exists in only one direction, it runs the *uni-ring* protocol.

When no ring exits, at least one link failed in each direction, so at least one segment of the ring is disconnected in the sense that it has no way of receiving messages (and recovering the clock) from nodes not in the segment (see Figure 5). In that case, the pre-designated leader may be disconnected. Thus in each segment the last node in the clockwise direction is selected as a *local* leader; this local leader runs the *chain* protocol.

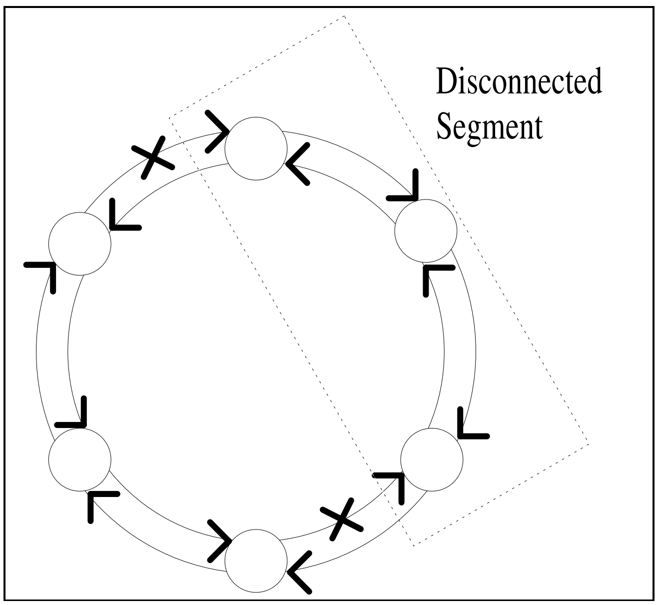

Figure 5: Disconnected Segment when no ring exists

3.2 The *State* Layer—The Ring Detection Sub-Protocol

The ring detection sub-protocol works in two parts, running concurrently. The purpose of the first part is to provide the leader with ring integrity indication in both the clockwise and the counter-clockwise directions; once this is known, the second part distributes this information among all other nodes. The code for the *State* layer is depicted in Figure 6.

To detect whether the ring is bidirectional, or if some link has failed in one direction or the other, the leader sets a bit (the *probe* bit of Figure 6) in all the frames sent in each direction. Each one of the other nodes, that detects a fault on an incoming link [3] in the

[3]SDH standards provide mechanisms (alarms) by which end nodes of a link detect its status, whether it

```
local variable:  CW_ring_status                    /* The computed clockwise ring status */
□
Do_for_ever
If the incoming link in the clockwise (CW) direction is down
    CW_ring_status := FAIL
    write FAIL into outgoing probe bit in the CW direction
elseif                                                        /* incoming link is up */
    if leader
        write OK into outgoing probe bit
        CW_ring_status := value of probe bit received from the CW direction
    elseif                                                    /* non-leader node */
        write received probe bit value into outgoing probe bit
        CW_ring_status := value of status bit received from the CW direction
write CW_ring_status into outgoing status bit in the CW direction
End Do_for_ever
□
```

Figure 6: Code of the Self-Stabilizing *State* Layer for the clockwise (CW) direction. The code for the counter-clockwise direction is identical.

clockwise (counter-clockwise) direction resets (to false) this bit in all the frames transmitted in the clockwise (counter-clockwise) direction; otherwise it relays the received bit value. If the bit returns to the leader set (true) then the ring operates correctly in the corresponding direction.

In the second part, an additional bit (the *status* bit of Figure 6) is used in the frame header. The leader sets this bit in each direction to indicate the status of the ring in that direction. Each one of the other nodes relays the received bit value if its incoming link is up, and otherwise resets the bit to indicate that the ring is faulty in that direction.

This simple lower-level protocol satisfies the following properties:

- The protocol requires only two bits in the frame header in each direction.
- The protocol is self-stabilizing.
- The worst-case stabilization time of the protocol is $O(|V|)$. This is optimal, since information about topological changes cannot propagate faster than $O(|V|)$ in a ring.

Proof of correctness, proof of the self-stabilization property and analysis of the stabilization time are given in Appendix B.

3.3 The *Reset* Layer

A fail-safe reset mechanism is required for initializing all nodes and re-starting the protocol when configuration changes occur, thus making the ring protocol self-stabilizing.

is operational or not. These mechanisms enable a node to detect a faulty outgoing link only if the parallel incoming link is operating.

The protocol presented here is a uni-directional version of the fail-safe global reset protocols of [1, 2, 13] adapted to the SDH abstraction model and based on the knowledge of the state of the ring, as detected by the *state* layer.

For each possible ring state (as reported by the state layer) we define a *reset cycle*, which passes through all the nodes in the ring (or in each strongly-connected component), and includes a *cycle leader*. In case one or both ring direction are operating, the cycle leader is the pre-designated leader, and the reset cycle is the operating ring (if both directions are operating, then the clockwise ring may be arbitrarily selected as the reset cycle). When no ring exists, the reset cycle runs from one end of each maximal connected segment to the other end and back, and one of the end nodes (the last node in the clockwise direction) is the cycle leader.

Any node that detects a change in the local configuration (either an input change, or a change in the ring topology) as detected by the lower level state layer, sends a reset request bit to the cycle leader. The leader "freezes" and then "de-freezes" the nodes in the cycle, using another bit in the frame header. This eliminates multiple reset commands and acknowledgments. The reset layer purges the values of all the variables of higher layer algorithms in a node when it "freezes" the node, and restarts the higher layer algorithm in its initial state when it "de-freezes" the node.

Note that this mechanism requires only two bits in the frame header in each direction.

Reset layer code

The code for the reset layer is depicted in Figure 7. Each node, except for the leader, may be in one of three states: NORM—normal operation, REQ—requesting a reset, or RST—"frozen." The leader may only be in the NORM or RST states. Each node continuously writes its state into the proper location in the out-going frames in the cycle direction, hence the state of each node is an input to its successor in the cycle. Since no more than three distinct states are defined, the transmitted state may be encoded using only two bits in the frame header. Upon entering state NORM, any node resets all the variables of its higher layer algorithms to their initial values and restarts these algorithms.

The code is described in a tabular form that describes a state transition function: For each node in a given state and and for any input (predecessor state, or a configuration change signal from the lower level state-layer, or a change in a node's input (e.g., connecting a new external clock source)), the table specifies the node's next state. The right half of the table specifies the code for the cycle leader, while the left half specifies the code for any other node.

This code satisfies the following properties:

- The protocol requires only a small, constant number of bits (two bits) in the frame header in each direction.
- The protocol is self-stabilizing.
- The worst-case stabilization time of the protocol is an optimal $O(|V|)$.

Proof of correctness and proof of the self-stabilization property are given in Appendix C.

	Non-leader state transition function				Leader state transition function			
if	Next leader state is				Next non-leader state is			
at state	Read NORM	Read REQ	Read RST	Config. Change	Read NORM	Read REQ	Read RST	Config. Change
NORM	NORM	REQ	RST	REQ	NORM	RST	NORM	RST
RST	NORM	REQ	RST	REQ	RST	RST	NORM	RST
REQ	REQ	REQ	RST	REQ				

Figure 7: Reset layer - code for Cycle Leader and for other nodes

3.4 The *Tree* Layer

We first describe three basic algorithms:

A. The Enumerating algorithm,

B. The Selection algorithm, and

C. The Quality Detection algorithm,

and then use these algorithms as building blocks in the construction of the protocol for the *tree* layer.

A different tree-layer protocol is constructed for each of the possible ring states (i.e., when a complete ring exists in both directions, when a complete ring exists in only one direction, and when no complete ring exists). Each of these ring-layer protocols uses a different combination of the three basic algorithms (see Figure 4).

In each node in each ring state, the corresponding tree-layer protocol is invoked by the reset layer when the reset layer enters the NORM state. Which of tree-layer protocol is used is determined by the output of the state layer. The tree-layer protocol repeats itself indefinitely in order to support self-stabilization.

Basic Algorithm A - Enumerating

Here, nodes are globally enumerated, i.e., each node is given a number equal to its distance from the initiator in a specific direction; the initiator also detects n, the total number of nodes. The algorithm works for any uni-directional ring.

Using ideas from [17] we design an enumerating protocol that works in $\log n$ rounds, using only four bits in the frame header (see Figure 8).

In each round, one bit in the binary representation of the enumeration is computed. In the first round, a circulating bit (the 'cur' bit in Figure 8) alternately marks the nodes even / odd, by each node writing into the corresponding bit in the header the negation of the bit its predecessor wrote into its incoming register. In each of the next rounds, the next least significant bit is set by the circulating bit. This bit only alternates at nodes that have accumulated a binary representation of all ones in previous rounds (e.g., in the fifth round, the bit alternates only at nodes that have built a binary representation of 1111 in their least significant four bits). Note that the number accumulated in the leader is n, the total number of nodes in the ring.

```
bits                                                   /* header bits used by the algorithm */
    enum { on    : bit                                 /* set when the algorithm is running */
           cur   : bit                               /* current bit in binary representation */
           round : bit                                        /* round differentiation bit */
           term  : bit                                       /* termination detection bit */
           }
□
variables
    next_bit : bit                                                    /* next bit to send */
    iter     : bit                                              /* next round bit to send */
    stop     : bit                                 /* next termination detect bit to send */
    done     : bit                                      /* termination flag for the leader */
    num      : integer                                                     /* accumulator */
□
if (Leader)
    iter := num := done := 0
    while (done = 0) do
        write (1, 0, iter, 0) into output buffer
        if (enum.on = 1) and (enum.round = iter)
            set next bit of num to enum.cur
            iter := 1 − iter                          /* to distinguish between rounds */
            if (enum.term = 1)
                write (0, 0, 0, 0) into output buffer
                done := 1
elseif                                                               /* not the leader */
    do forever
        if (enum.on = 0)
            if (num ≠ 0) then output num
            iter := num := 0
            write (0, 0, 0, 0) into output buffer
        elseif
            if (enum.round = iter)                               /* is this a new round? */
                set next bit of num to enum.cur
                if (num is all ones) or (num is empty)
                    next_bit := 1 − enum.cur
                    stop := 1
                elseif
                    next_bit := enum.cur
                    stop := enum.term
                write (1, next_bit, iter, stop) into output buffer
                iter := 1 − iter
□
```

Figure 8: Code of the Enumeration Algorithm. Note that this code runs on top of the *reset* layer, which makes it fail-safe and self-stabilizing.

The nodes differentiate between consecutive rounds by an additional circulating bit (the 'round' bit in Figure 8) that is glued to the 'cur' bit. The leader toggles this bit when starting a new round. Each of the other nodes relays this bit's value to the next node down the ring, and detects the beginning of a new round when it detects a change in this bit.

The leader detects the termination of the process by using a third circulating bit (the 'term' bit in Figure 8). This bit is reset by the leader, but is set by any node that alternates the 'cur' circulating bit. Thus, the leader detects termination only when it detects that this bit was not set after completing a full round, so no more bits in the binary representation of n need to be computed.

As an option, and using one additional bit, the number of nodes in the ring, n, may also be distributed to all the nodes. This additional bit conveys, in each round, the last bit in the binary representation of n, as computed by the leader in the previous round. The most significant bit of n is conveyed to the nodes in the last round in which termination is detected.

When, due to link failures, a complete ring exists in only one direction, parts of the ring may still be bi-directional. In that case, in addition to the global enumeration of the nodes in the ring, we also wish to enumerate the nodes *locally*, i.e., to enumerate the nodes within each such bi-directional part (see Figure 9). Such local enumeration may optionally be done using one more circulating bit in the frame header. This bit is used in a similar manner as the 'cur' circulating bit (see above), except that it is always initialized at the beginning of each bi-directional part.

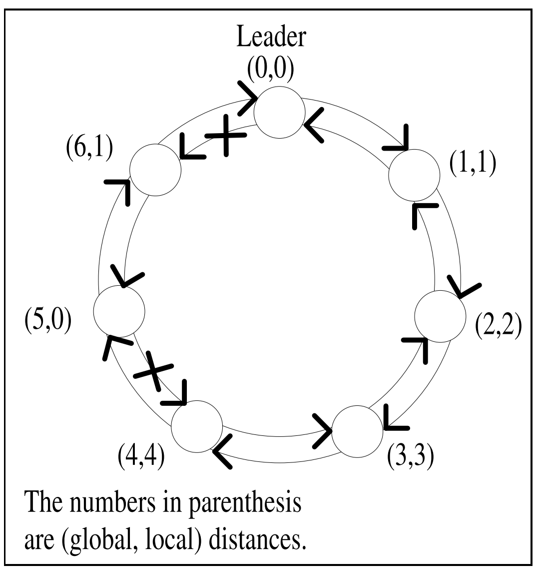

Figure 9: Local Numbering

Basic Algorithm B - Selection

Suppose that nodes are given integer grades, in the range $[1..n]$. This algorithm finds the highest-grade node, that owns a specific quality, and sets it to be the source.

The algorithm works for any uni-directional ring in $\log n$ rounds, using four bits in the frame header.

In each round, one bit in the binary representation of the grade is set, starting from the most significant bit. In the first round, the leader detects if the specified quality is available within the nodes having a one in the most significant bit of the binary representation of their given grades, using a circulating token (as in Basic Algorithm A above). In the next round, this bit is broadcast to all nodes (using a second circulating bit), and the leader detects if the specified quality is available among the remaining nodes that have a one in their next most significant bit. In the last round, the source is set.

Basic Algorithm C - Search for Best Quality

Here, the best quality available is detected by the leader.

The leader sends a QUERY(Q) message,[4] containing the leader's quality Q. In the dual-ring case, the message is sent in one of the directions (say the clockwise direction). In the uni-ring case, the message is sent in the operating ring direction. In the chain case, the message is sent towards the other end of the segment. Each node, upon receiving this message, compares it to its own quality. If its own quality is better, it replaces Q in the message with its own quality and forwards the message; otherwise it forwards the message as is. In the chain case, the node at the other end of the segment returns the message back to the leader's direction. After one round, the leader knows the best quality available in the ring.

3.4.1 The Dual-Ring Protocol

This is the protocol that is invoked by the reset layer when the ring was detected to be fully operational in both directions.

First, the leader determines Q, the best quality available in the ring, by running Basic Algorithm C.

Second, the leader arbitrarily selects the source, i.e., the node that would root the timing distribution tree. To this end it sends a GRANT(Q) message, and the first node of quality Q to capture the message is selected as the source.

In the third step, the selected source forms a balanced timing distribution tree:[5] it runs Basic Algorithm A (enumeration), with the option of distributing n (the number of nodes). Then each node, knowing its own distance from the source and the total size of the ring, locks itself to the direction of the shorter path to the source.

3.4.2 The Uni-Ring Protocol

This is the protocol that is invoked by the reset layer when the ring operates in only one direction.

[4]According to current standards, the number of possible quality levels is a small constant, that may be represented using only four bits. If there are many quality levels, a $\log |Q|$ algorithm similar to Basic Algorithm B may be used.

[5]or, in the case of an even number of nodes in the ring, an almost balanced tree, with a difference of one between the lengths of the two branches.

In that case, parts of the ring may still be bi-directional. For the sake of protocol simplicity, if the pre-designated leader is not the first node on such a bi-directional part, the leader grants the first next such node down the ring to be the new leader (by setting a GRANT bit, that is captured by that node).

The new leader first detects the best quality available in the ring by running Basic Algorithm C.

The leader then selects the source. In this case, the source cannot be selected arbitrarily among the nodes offering the best quality in the ring; it should be the one that forms the most shallow spanning tree among them. Note that if the protocol selects, among the best-quality nodes, the one with the longest "tail" (i.e., the longest path of connected nodes) in the direction *opposite* to the operating ring direction, then the tree rooted at this node is the required most shallow tree. Let us clarify this by an example: in the left side of Figure 10, two links failed in the inner ring, so two bi-directional segments are formed. If nodes x and y own the best quality clocks, node x should be selected as the source, because it roots a tree of a lesser depth than any tree rooted by node y. The resulting timing distribution trees rooted by nodes x and y are depicted in the right side of Figure 10.

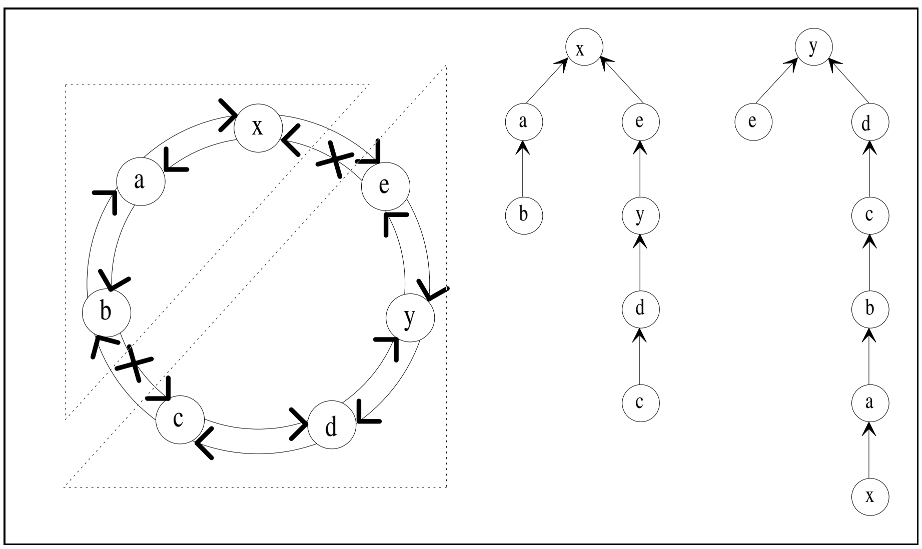

Figure 10: Uni-Ring Case: Sources and Resulting Trees

Thus the leader selects the best-quality node that has the longest "tail" as the source. To do this, it first runs Basic Algorithm A (enumeration) with both the option for local enumeration within bi-directional parts, and the option for distributing n. Next, the leader selects the source by running Basic Algorithm B, in which the grades given to nodes are their *local* numbers.

In the third step, the source forms the timing distribution tree. After Basic Algorithm A has terminated in the previous step, both n and the "tail" length are known to the source. The source compares its "tail" length to $n/2$.

If its "tail" is less than $n/2$, a balanced tree cannot be formed. In that case, the most shallow tree is constructed by forming one path that includes all the nodes in the "tail", and a second path that includes all the other nodes. Thus the source locks all the nodes to itself in a straight-forward manner, sending simple LOCK messages first in the direction of the "tail", and then in the opposite direction.

If the "tail" length is greater than $n/2$, a balanced tree may be formed[6], and not all the nodes in the "tail" share the same path in the tree. Thus, in this case the source builds a balanced tree in a way similar to the dual-ring protocol: it re-runs Basic Algorithm A (enumeration) with the option for distributing n. Then all the nodes, according to their own numbers and n, decide on the direction to lock to.

3.4.3 The Chain Protocol

This is the protocol that is invoked by the reset layer when no complete ring exits in either direction. In this case, the ring is partitioned into independent, fully-connected bi-directional chains. The *chain* protocol is run in each chain independently.

In each such chain, a local source is selected and all nodes in the chain lock their timing circuits to the clock of that local source. In each chain, the source should be chosen such that its quality is the best quality available in the chain, and such that it is as centered as possible within the chain.

Since the pre-designated ring leader may not exist in the chain, the protocol is run by a *local* leader, defined as the last node in the chain in the clockwise direction. In all of the following steps, the local leader runs each "building block" on a uni-directional ring formed by the two end nodes closing a U-turn connection between the opposite directions.

The local leader first detects the best quality available in the chain by running Basic Algorithm C[7].

Next, the local leader runs Basic Algorithm A, during which, each node computes two distances, one from each end of the chain.

Now the nodes re-number themselves, such that the higher number nodes are in the middle of the chain. This new numbering must not necessarily be unique. To do this each node selects the minimum among the two computed distances.

The source is selected by the local leader running Basic Algorithm B (selection), in which the grade given to each node is that minimum.

The source locks all other nodes to itself by sending simple LOCK messages in both directions simultaneously.

3.5 Using the Ring Protocol for Other Simple Topologies

The ring protocol is also suitable for use in the other basic SDH/SONET topologies. The protocol may be used as-is in the uni-ring and chain topologies, since these are treated as special cases of failures of the dual-ring topology.

The protocol may trivially be expanded to be used in ring networks with *appendages* (an appendage, in SDH terminology, is a single node connected to a topology via a single pair

[6] or, again, if the number of nodes is even, an almost balanced tree.

[7] The nodes at the ends of the chain may have operating incoming links from outside the chain. These nodes must include those links as valid timing sources in order to use the best possible quality in the chain, and to construct the largest spanning tree possible in the network.

of anti-parallel links): the only modification required is for a node that has an appendage to include the appendage's timing sources within its own.

A trivial protocol may be used for the star topology. In this protocol, the star center is the leader, and the protocol, like the dual-ring protocol, has three steps: find the best quality offered within the star, select the source, and build the distribution tree. All of these steps are trivial in the star topology.

4 Protocols for General Topologies

The above presented protocols for the ring topology cannot be easily adapted to general topologies, since it heavily relies on the ring properties. One might think of generalizing the ring protocol by first finding an Euler cycle in the network (using the technique presented in [18]), and then running the ring protocol over this cycle. However, the shortest paths found in the Euler cycle are not guaranteed to be the shortest paths in the underlying graph. Hence, other solutions are required for general-topology networks.

4.1 Bi-directional Networks

In this section we propose protocols for general-topology networks which are bidirectional, i.e., for each operating link from u to v there exists an operating link from v to u.

4.1.1 A Timing Forest for Bi-directional Networks

We propose the following self-stabilizing simple protocol for constructing a timing *forest* in a general topology bi-directional network. The presented protocol is an application of the self-stabilizing network protocol techniques presented in [9, 19, 1, 20, 2, 11, 21, 8, 4]. The protocol does not require that unique IDs are given to the nodes, nor does it require a leader, however we assume the knowledge of some upper-bound, D, on the network diameter.

We define two registers in each node (which correspond to two fields in the frame header): one (the *qual* variable of Figure 11) holds the quality owned by the root of the node's tree, and the other (*dist*) holds the node's distance from that root. Each node also marks the link that leads to the root (using the variable *anc* in Figure 11). Each of the nodes continuously writes the values of its registers to its neighbors, and continuously compares its registers with those of all of its immediate neighbors (extracted from incoming frames)[8]; if a path to a better-quality root, or if shorter path to an equal-quality root is found, then the node marks the link that leads to the new root, and updates its registers accordingly. When an illegal state is detected by a node, the node initializes its registers such that it sets itself to be a root: the distance register is reset to zero, the quality register is set to the node's own quality, and no link is marked. Note that the *dist* variable is responsible for forming the most shallow possible trees, but also serves to eliminate timing loops.

The formal code is depicted in Figure 11. Proof of correctness and of the self-stabilizing property is given in Appendix D.

[8]Since our model requires fixed-length messages, the binary representation of the distances may be grouped into fixed-size units (e.g., nibbles or octets), and the comparison may be done using those units.

```
local variables
    own_q /* node's own best self-owned clock quality,
             in non-volatile memory */
    qual  /* root's quality */
    dist  /* distance to root */
    anc   /* link to ancestor in tree */
    α     /* list of in-coming links, maintained
             by a lower-layer self-stabilizing protocol */
□
do forever
    q := max_{l∈α}{qual value read from incoming link l}        /* Compare registers; */
    d := min_{l∈α}{dist value read from any incoming link l
                  whose qual value is q}
    if (q>qual) or (q=qual and d+1<dist)                      /* If a better path found, */
        qual := q                                             /* then update registers. */
        dist := d + 1
        anc  := any l ∈ α whose qual value is q and dist value is d
    if (own_q≥qual) or                                        /* If own quality is better, or */
      ¬((anc=nil and dist=0              and qual=own_q) or    /* if illegal state, */
        (anc≠nil and dist= 1+anc.dist and qual=anc.qual) or
        (dist< D))
        qual:= own_q                                          /* then initialize registers. */
        dist := 0
        anc  := nil
                                      /* Write variables into out-going links to neighbors */
    write qual, dist into all out-going links
□
```

Figure 11: Code of the Self-Stabilizing BFS Forest Protocol for Bidirectional General-Topology Networks.

4.1.2 A Timing Tree for Bi-directional Networks

We propose the following outline for establishing a single timing distribution tree in a bi-directional general-topology network, using standard building blocks from the field of distributed computing:

1. The leader initializes a search of the network (as in [22]), in which the best quality available is detected.

2. The leader initializes a center-finding process, as described in [23][9].

3. The leader passes control to one of the nodes in the graph center.

4. That node starts a BFS search of the graph (see [24]), searching for the closest node offering the best quality in the network (as detected in the first step). This node is set to be the root.

5. The root constructs a BFS tree in the network to lock all the other nodes to itself through the shortest paths.

Changes in the configuration require a global reset, as described in [13].

Note that the center-finding process is very expensive in terms of time and message complexity ($O(n|E|)$ message complexity). This is due to the fact that in [23], each node v calculates $r(v)$, which is the maximum distance of any other node to v. Thus, we may use one of the following ways to reduce the number of nodes that calculate $r(v)$:

1. Only the nodes offering the best-quality clock sources take part in the center-finding process (i.e., only those nodes calculate $r(v)$). Since the number of nodes offering the best-quality (atomic) clock source in a real network is small, this considerably reduces the time and message complexity.

2. A probabilistic method: each node tosses a coin, and decides whether or not to calculate $r(v)$.

3. Partitioning the graph: forming neighborhoods of n/Δ nodes. In each such neighborhood only one node calculates $r(v)$. This reduces the number of calculations, while producing a reasonable approximation to the center. Note that in the real-world timing distribution problem, a timing distribution path that is almost as short as the shortest path is acceptable, and hence, finding the exact center would not be critical in most applications.

5 Conclusions and Further Research

We have defined a new model for abstracting SDH/SONET networks using constant length, single-reader, single-writer shared registers. We then used the model to provide synchronization algorithms that construct a timing tree (or a forest of timing trees) in a network.

[9] If the requirement for minimal tree depth is relieved, then this step is not necessary.

A detailed tree-constructing protocol was presented for the ring topology. We also outlined protocols for the other basic topologies common in simple SDH/SONET networks, and described protocols for general-topology bi-directional networks.

Further research is required to find an efficient implementation of synchronization protocols for general-topology uni-directional networks. A special effort is required for implementing a fail-safe reset process and/or self-stabilizing protocols in arbitrary topology strongly-connected uni-directional networks.

As SDH networks steadily increase in size, *central* network management systems will find it harder to control the networks and to provide the system administrators with a proper quality of service (e.g., response time). The size increase means long delays, an enormous amount of management information, and heavier loads on the limited-bandwidth management communication channels (see Appendix A). Thus, *distributed* protocols might be called in to handle some of these tasks, such as:

- topology maintenance,
- bandwidth allocation,
- automatic, reliable routing of management information,
- automatic alternative routing in cases of failures of links and nodes in the network, and
- fault detection and isolation.

Acknowledgments

We are thankful to Anat Bremler for pointing out an error in Section 4 in an earlier version of the paper. We would like also to thank Noga Alon for a useful discussion and to thank the referees of INFOCOM-96 for many helpful remarks that have improved the presentation. This work was supported by the Broadband Telecommunications R&D Consortium administered by the Chief Scientist of the Israeli Ministry of Industry and Trade and by ECI Telecom Ltd.

References

[1] B. Awerbuch, , and G. Varghese. Distributed program checking: a paradigm for building self-stabilizing distributed protocols. In *Proc. of the 32nd IEEE Ann. Symp. on Foundation of Computer Science*, pages 258–267, October 1991.

[2] B. Awerbuch, B. Patt-Shamir, and G. Varghese. Self-stabilization by local checking and correction. In *Proc. of the 32nd IEEE Ann. Symp. on Foundation of Computer Science*, pages 268–277, October 1991.

[3] A. Mayer, Y. Ofek, R. Ostrovsky, and M. Yung. Self-stabilizing symmetry breaking in constant space. In *Proc. 24th ACM Symp. on Theory of Computing*, May 1992.

[4] G. Varghese. *Dealing with Failure in Distributed Systems.* PhD thesis, MIT, Department of Electrical Engineering and Computer Science, 1992.

[5] E. W. Dijkstra. Self-stabilizing systems in spite of distributed control. *Communication of the ACM*, 17:643–644, November 1974.

[6] S. Dolev, A. Israeli, and S. Moran. Uniform dynamic self-stabilizing leader election. In P.G. Spirakis S. Toueg and L. Kirousis, editors, *Lecture Notes in Computer Science 579: Proceedings of the fifth Int. Workshop on Distributed Algorithms*, pages 163–180. Springer Verlag, October 1991.

[7] A. Israeli and M. Jalfon. Token management schemes and random walks yield self-stabilizing mutual exclusion. In *Proc. 9th ACM Symp. on Principles of Distributed Computing*, pages 119–132, August 1990.

[8] S. Katz and K. J. Perry. Self-stabilizing extensions. *Distributed Computing Journal*, 7, 1994. Abstract In *Proc. 9th ACM Symp. on Principles of Distributed Computing*, August 1990.

[9] S. Dolev, A. Israeli, and S. Moran. Self-stabilization of dynamic systems assuming only read/write atomicity. *Distributed Computing Journal*, 7, 1994. also In *Proc. 9th ACM Symp. on Principles of Distributed Computing*, August 1990.

[10] Y. Afek and G. M. Brown. Self-stabilization over unreliable communication media. *Distributed Computing Journal*, 7:27–34, 1993. Abstract in *Proc. of the 8th IEEE Symp. on Reliable Distributed Systems*, 10-12 1989.

[11] Y. Afek, S. Kutten, and M. Yung. Memory-efficient self stabilizing protocols for general networks. In *Proc. of the 4th Int. Workshop on Distributed Algorithms*, September 1990.

[12] G. M. Brown, M. G. Gouda, and C-l Wu. Token systems that self-stabilize. *IEEE Transactions on Computers*, c38(6):845–852, 1989.

[13] Y. Afek, B. Awerbuch, and E. Gafni. Applying static network protocols to dynamic networks. In *Proc. of the 28th IEEE Ann. Symp. on Foundation of Computer Science*, pages 358–370, October 1987.

[14] B. Awerbuch and S. Even. Reliable broadcast protocols in unreliable networks. *NETWORKS*, 16:381–396, 1986.

[15] I. Cidon and I. Gopal. PARIS: An approach to integrated high speed private networks. *International Journal of Digital and Analog Cabled Systems*, 1(2):77–86, April-June 1988.

[16] I. Cidon, I. Gopal, and S. Kutten. New models and algorithms for future networks. In *Proc. of the 7th ACM Symp. on Principles of Distributed Computing*, August 1988.

[17] Y. Afek and E. Gafni. Distributed algorithms for unidirectional networks. *Siam J. on Computing*, 23(6), December 1994.

[18] Y. Ofek and M. Yung. Principles for high speed network control. In *Proc. 9th ACM Symp. on Principles of Distributed Computing*, pages 161–175, August 1990.

[19] A. Arora and M. Gouda. Distributed reset. In *Proc. of the 10-th FSTTCS: Springer-Verlag LNCS 472*, pages 316–331, September 1990.

[20] S. Dolev, A. Israeli, and S. Moran. Uniform dynamic self-stabilizing leader election. In P.G. Spirakis S. Toueg and L. Kirousis, editors, *Lecture Notes in Computer Science 579: Proceedings of the fifth Int. Workshop on Distributed Algorithms*, pages 163–180. Springer Verlag, October 1991.

[21] W. P. Tajibnapis. A correctness proof of a topology information maintenance protocol for a distributed computer network. *CACM*, 20-7:477–485, 1977.

[22] A. Segall. Distributed network protocols. *IEEE Trans. on Information Theory*, IT-29(1):23–35, January 1983.

[23] E. Korach, D. Rotem, and N. Santoro. Distributed algorithms for finding centers and medians in networks. *ACM Trans. on Programming Languages and Systems*, 6(3):380–401, July 1984.

[24] R. G. Gallager. Distributed minimum hop algorithms. Technical Report LIDS-P-1175, M.I.T. Lab for Information and Decision Systems, January 1982.

[25] ITU-T G.707, G.708, G.709, G.781, G.782, G.783, G.784, G.803, G.825.

[26] ETSI DE/TM-1015 Transmission and Multiplexing (TM): Generic Functional Requirements for SDH Transmission Equipment, parts 1 through 7.

[27] ANSI American National Standards for Telecommunications T1.102-1987, T1.105-1991, T1.107b-1991.

[28] ITU-T G.708, G.709, G.783

[29] Transmission and Multiplexing (TM): Generic Functional Requirements for SDH Transmission Equipment, Part 6: Synchronization Distribution Functions, ETSI DE/TM-1015-6, version 0.4, April 1994

[30] Transmission and Multiplexing (TM): Generic Requirements for Synchronization Networks, ETSI DE/TM-3017 parts 1-5, version 0.1, October 1993

[31] A Technical Report on Synchronization Status Messages, document T1X1.3/93-001R2 prepared by T1X1.3 Working Group on Synchronization and Tributary Analysis, ANSI 1993

A SDH Format, Management and Protocol Levels

A.1 Background

Digital transmission systems have provided steadily increasing bandwidth during the past couple of decades. More bandwidth became available when technology permitted transmission at higher speeds. These digital systems were based on the technology available at the time this development began. The result was the Plesiochronous Digital Hierarchy (PDH).

PDH evolved into higher system speeds as transmission technology developed. Higher speeds were incorporated as add-ons for lower-speed systems. This proved satisfactory for some time, as these kinds of systems were adequate for the applications, taking the cost of the systems into consideration.

However, there have been several developments in recent years: new demands for transmission features have arisen; higher bandwidth has become technologically feasible; and cost, relative to the transmission capacity provided, has decreased.

As a result, a new system appropriate to the new requirements has been defined, the Synchronous Digital Hierarchy (SDH). This system has a number of advantages compared to PDH systems:

- SDH provides simpler multiplexing. An SDH signal has lower-level signals (i.e., lower signal rates) embedded—as in PDH systems. However, the lower SDH levels can be directly identified from the higher level. This makes drop-and-insert and cross-connect of embedded traffic channels in the SDH system much simpler than in PDH systems.

- SDH provides an optical standard.

- SDH is prepared for future transmission technology: the definition of SDH for higher-speed transmission rates is very straightforward.

- SDH is prepared for future applications. Current SDH standards focus on carrying existing PDH signals and ATM signals, but the definitions are open for the inclusion of other applications, such as HDTV and MANs.

- SDH provides built-in signal capacity for advanced network management and maintenance capabilities. Data channels for operation and maintenance of the SDH network are embedded within the SDH signal itself, and are therefore available at SDH network elements.

- The SDH standards include a computer-based, integrated network management and maintenance system.

- SDH unifies the three traditional network application areas: Local Network, Inter-exchange Network and Long Haul Networks into one integrated system, managed by one manager entity.

A.2 SDH Network Structures

Today, four levels of SDH signals have been defined. The basic SDH rate, STM-1 (Synchronous Transfer Module level 1) is defined at 155.52Mbits/sec. The other rates defined are STM-4, STM-16 and STM-64, which are the appropriate multiples of the basic bit rate. These rates are used in networks, depending on bandwidth requirements. Typically, the network core (or backbone) uses links at the higher rates, while at the regional and local levels, the lower rates are used.

Typically, ring structures are used in the local and regional levels to provide for protection of the traffic, while point-to-point links are used in the core network.

A.3 SDH Format and Overhead Bytes

The SDH synchronous signal comprises a set of 8-bit bytes which are organized into a frame structure. Within this frame structure, the identity of each byte is known and preserved with respect to a framing (or marker) byte.

The SDH frame consists of payload and overhead bytes, and is repeated 8000 times per second. Hence the basic SDH rate, STM-1, is 2430 bytes per frame, at a rate of $2430 \times 8 \times 8000 = 155$ Mbit/second. The payload bytes carry the application transmission carried by the network. The overhead bytes are used to carry control, alarm, performance monitoring and management information. Each one of the overhead bytes, designated by its offset from the framing byte (i.e., the start of the frame), is used for a specific function; some of these overhead bytes are reserved for future applications. Each such byte is repeated in every frame.

A.4 SDH Management

One of the advantages of SDH compared with existing PDH systems is that the SDH networks function together with a standardized, centralized operation and maintenance system. The management system is composed of several layers.

Each node is controlled by a local processor using local memory. This processor collects status information from the node's hardware, and configures the node by direct control over the hardware.

The Network Manager (NM) is a work station with network management software, connected to one of the nodes in the network and communicating with all other nodes via a virtual dedicated Data Communication Channel (DCC). This channel is implemented between any two neighboring nodes using some of the overhead bytes as the low-level "physical" link, and a level-2 protocol (such as HDLC) above that; up to 12 of the overhead bytes in each frame are used to form a DCC channel, at a maximum rate of $12 \times 8 \times 8000 = 768$ Kbits/second. This channel is used to convey alarm, control, maintenance and performance monitoring information between the nodes' local processors and the Network Manager.

The functions covered by the NM central manager are:

- Fault (maintenance) management: alarm surveillance, and testing.
- Performance monitoring (e.g., collection of bit error performance).

- Configuration management. This includes topology management (e.g., adding and removing managed network elements), configuration of all equipment and transmission attributes, trail management (definition of payload connections) etc.
- Security management (e.g., users and permissions).

The NM is the highest level of control and can thus be used for running centralized algorithms and for controlling parameters within each node in the network. However, the NM's operation depends on the proper and stable operation of many network components. Therefore, the NM may be used to run static algorithms in the network, but is not suitable to run dynamic algorithms for cases of failures in the network.

A.5 Network-level Protocols in SDH

A.5.1 Levels of Protocols

Network-level protocols in SDH may thus run in one of two levels. The higher level is the Network Management level. In this level, the algorithms are run by management entities, and inter-node messaging are via the DCC channels. Protocols and algorithms using this level may be complex and may use complex, long messages. However, most of the protocols of this level require that the network is operating in full and that the management entities (which differ from the data traffic entities) are running and properly configured. This is not required by the lower level. Protocols in the lower level run locally within the processors in each node and rely on simple, constant-length messages in the SDH frame's overhead bytes. Protocols in this level also relieve some load off the management processors and free precious bandwidth on the DCC channels.

A.5.2 Hardware Implications

Some hardware in each node is dedicated for generation (in the transmit direction) and termination (in the receive direction) of overhead bytes. Some of the overhead bytes are calculated by hardware operations; for example, parity bytes (used for error performance monitoring) are calculated by the hardware, since this is a simple task yet too fast for any common processor (remember that the cycle time for a single byte, in the *basic* SDH rate, is about 50 nano-second). Most other bytes are *visible* to the software: these bytes are written by the software into hardware registers and are transmitted by the hardware in the transmit direction; in the receive direction, the values received by the hardware is written into registers readable by the software.

The software's processing time may be much longer than the hardware's frame time. So in the transmit direction, each overhead byte written by the software is re-transmitted every frame by the hardware. In the receive direction, each received overhead byte over-writes the same byte received in the previous frame, even if the software did not read the previous value. Protocols designed to use these bytes must take this into account.

On the other hand, any requirement for "real-time" performance of a protocol (e.g., frame counting) requires additional hardware. Adding hardware has the drawbacks of cost, power and heat dissipation, hardware development cycle time, and complicated upgrade of existing equipment.

Therefore, new low level protocol suggested should not pose new "real-time" requirements, and should use stable messages (i.e., messages that stay stable for the time required for the receiving node to detect them). Any protocol designed to meet these requirements works in existing networks.

B The State layer - proofs

In this section, we give proof of correctness, proof of the self-stabilization property and stabilization time analysis for the State layer of the Ring protocol.

B.1 Proof of correctness

Lemma 1 *A finite time after any change in the links' status, the* clockwise_ring_status *variable in the leader is OK iff all the links in the clockwise direction are operational.*

Proof. If all the links in the clockwise direction are operational, then (by induction on the distance of nodes from the leader) all nodes eventually read and write OK in the *probe* bit. In particular, the node nearest to the leader in the counter-clockwise direction writes OK in the *probe* bit on the link to the leader, so the leader sets *clockwise_ring_status* to OK.

If the link incoming directly to the leader has failed, immediately the leader sets *clockwise_ring_status* to FAIL. Else, observe the failing link that is closest to the leader in the counter-clockwise direction. The node this link enters detects the failure and writes FAIL in its outgoing *probe* bit in the leader direction. This bit essentially propagates through the remaining nodes down to the leader, and the leader sets *clockwise_ring_status* to FAIL.
□

Lemma 2 *A finite time after any change in the links' status,* clockwise_ring_status *in each node is OK iff all the links in the clockwise direction are operational.*

Proof. The proof uses induction on the distance of nodes from the leader in the clockwise direction.

For the first node, if the link incoming from the leader is faulty, then this node rightfully sets its *clockwise_ring_status* variable to FAIL. Else, the node sets this bit to the the read *status* bit coming from the leader, which determines whether all the links in the clockwise direction are operational (as shown in the previous lemma).

For the next nodes, if the link from the previous node is faulty, then *clockwise_ring_status* is rightfully set to FAIL. Else, the node sets this bit to the *status* bit coming from the previous node, which (from the induction assumption) determines if all the links in the clockwise direction are operational.
□

Exchanging "clockwise" with "counter-clockwise" in the above lemmas (and their proofs), we get:

Lemma 3 *A finite time after any change in the links' status,* counter-clockwise_ring_status *in each node is OK iff all the links in the counter-clockwise direction are operational.*

B.2 Self-stabilization

Since in the above lemmas we have not assumed any variable initialization or any other initial condition, then:

Corollary 4 *The State layer is self-stabilizing.*

B.3 Stabilization time

Note that the leader's *ring_status* variables stabilize in at most $|V|$ time units after the last topological change, and those of the other nodes stabilize after at most $|V|$ more time units. Hence, the worst-case stabilization time of the State layer is $O(|V|)$. This is optimal, since information about topological changes cannot propagate faster than $O(|V|)$ in a ring.

C The Reset layer - proofs

In this section, we give proof of correctness and proof of the self-stabilization property for the Reset layer of the Ring protocol.

C.1 Proof of correctness

Observe the reset cycle (see the Reset protocol definition above). The cycle configuration may be described by a vector of the nodes' states. Let P be the set of all possible configurations, $q \in P$ be the configuration where $\forall v \in V$, $\text{state}_v = \text{NORM}$, and $r \in P$ be the configuration where $\forall v \in V$, $\text{state}_v = \text{RST}$. For states i and j in P, we denote $i \leadsto j$ (read "i leads to j") if, when starting from configuration i, the protocol essentially brings the cycle to configuration j.

Lemma 5 *After the last configuration change, for any initial state $p \in P$, $p \leadsto q$.*

Proof. The cycle leader cannot stay forever in a RST state, since then it will forever write RST to its successor, which will eventually write RST to its successor and so forth, until the cycle leader will read RST on its input and change its state to NORM.

The cycle leader cannot toggle forever between NORM and RST either, because then its predecessor in the cycle has to toggle forever between RST and REQ, which means that its predecessor has also to toggle forever between RST and REQ, and so on—which means the leader itself must be in the REQ state—contradiction.

Hence, the cycle leader eventually stabilizes in the NORM state. But it can be easily verified that this is only possible if the state of all the other nodes in the cycle is also NORM. □

Lemma 6 *Starting from initial state q, any configuration change causes the network to reach state r.*

Proof. Let us shorthand N for NORM, R for RST, and Q for REQ. Then a network configuration may be written as a string of length $|V|$ of the form $\{N^*R^*Q^*\}^+$.

If the configuration change was detected by the leader, then it may be easily verified that the following configurations follow: $q \equiv N^{|V|} \to RN^+ \to R^+N^* \to R^{|V|} \equiv r$; and if the configuration change was detected by a non-leader node, then the following configurations follow: $q \equiv N^{|V|} \to N^+QN^* \to N^+Q^+ \to RN^*Q^+ \to R^+Q^+ \to R^{|V|} \equiv r$.
□

Lemma 7 *Let q' be the state immediately following q after a configuration change was detected. Then any additional configuration change during the state transition $q' \rightsquigarrow r$ does not prevent the network from reaching r.*

Proof. It may easily be verified that for each of the configurations on $q' \rightsquigarrow r$ (as listed in the proof of the previous lemma), a configuration change also leads to r.
□

C.2 Self-stabilization

We have shown that, starting from any initial configuration, the network becomes quiescent, and that a configuration change (or multiple configuration changes) in the network cause all nodes to reset and then resume (since, if $\forall p \in P$, $p \rightsquigarrow q$ then particularly $r \rightsquigarrow q$). Hence,

Corollary 8 *The Reset layer protocol is self-stabilizing.*

D A timing forest for bi-directional networks - proofs

In this section, we give proof of correctness and proof of the self-stabilization property for the protocol that creates a timing forest in general-topology bi-directional networks.

We first define a network configuration which is "safe" (as in [9]), such that any fair execution[10] of the protocol, starting from this configuration, belongs to the set of *legitimate* sequences. In other words, we define a network configuration which expresses a correct network behavior, and is stable. Next we show that starting from any initial condition, the network eventually reaches a "safe" configuration as we have defined.

Lemma 9 *Let $Q = \max_{v \in V}\{own\_q_v\}$. Let c be a network configuration which satisfies:*

1. $\forall v \in V$, $qual_v = Q$.
2. $\forall v \in V$, $own\_q_v = Q$, $anc_v = nil$ *and* $dist_v = 0$.
3. $\forall v \in V$, $own\_q_v < Q$, anc_v *points to* u, $(u, v) \in E$, *such that* $dist_u = \min_{w,(w,v) \in E}\{dist_w\}$ *and* $dist_v = 1 + dist_u$.

Configuration c is safe for the protocol.

[10] An infinite execution is *fair* if every processor executes steps infinitely often.

Proof. It is not hard to verify that following configuration c, the value of no register ever changes. Hence, all subsequent configurations encode a valid forest of BFS trees, rooted at best-available-quality roots (which is a legitimate network state).
□

Lemma 10 *Let $E = c_0c_1c_2\ldots$ be an arbitrary fair execution of the protocol, and let $Q = \max_{v\in V}\{own\_q_v\}$ as above. Then:*

1. *For any configuration c_{i_1} where $\exists v \in V$, $qual_v > Q$, there exists a configuration c_{i_2}, $i_2 > i_1$, where $qual_v \leq Q$. Therefore, there also exists a configuration c_{i_3}, $i_3 \geq i_2$, where $\forall v \in V$, $qual_v \leq Q$.*

2. *For any configuration c_{i_3} where $\forall v \in V$, $qual_v \leq Q$, and $\exists u \in V$, $qual_u < Q$, there exists a configuration c_{i_4}, $i_4 > i_3$, where $qual_u = Q$. Therefore, there also exists a configuration c_{i_5}, $i_5 \geq i_4$, where $\forall v \in V$, $qual_v = Q$.*

3. *For any configuration c_{i_5} where $\forall v \in V$, $qual_v = Q$, and $\exists u \in V$, $own\_q_u = Q$, but ($anc_u \neq nil$ or $dist_u > 0$), there exists a configuration c_{i_7}, $i_7 > i_5$, where $\forall v \in V$, $qual_v = Q$ and $\forall u \in V$, $own\_q_u = Q$, $anc_u = nil$ and $dist_u = 0$.*

4. *For any configuration c_{i_7} where $\forall v \in V$, the first two assertions of the previous lemma hold, but the third assertion does not, there exists a configuration c_{i_9}, $i_9 > i_7$, which is safe.*

Proof.

1. In node v, where $qual_v > Q$, if $anc_v = nil$ then an illegal state is detected in the next execution step of v, and $qual_v$ will be set to own_q_v (which is $\leq Q$ by definition). Else, there exists a *path* consisting of nodes with equal *qual* variables (we define a path as a set of nodes where each node points to its successor in the path). Any fair execution of the protocol is guaranteed to break any cycle in this path (due to the use of the *dist* variable). Therefore, this path is eventually simple and finite. The last node on the path, u, either points to itself or to another node with a different *qual* variable. In any case, in its next execution step it initializes its registers and sets its *qual* variable to $own\_q_u \leq Q$. Following that, all the other nodes in the path eventually set their *qual* variables to values $\leq Q$.

   This does not prevent a large quality, q from rotating around a cycle and increase the distance parameter of the nodes without bound. In such a scenario, there is a sequence of nodes whose *qual* equals to q and while the node at one end of the sequence notices the inconsistency and resets, the node at the other end captures one more node and sets its quality to q with a higher distance parameter. In this way the sequence of nodes marked by q is moving around in a cycle, with higher and higher distance parameters. Following Gouda and Multari [19], this problem is avoided in our algorithm by using the knowledge of an upper bound on the network diameter. Any such rotating quality q would eventually reach a distance parameter higher than D and would disappear from the network, since the smaller distance nodes notice the inconsistency and reset them-self.

2. Since the network graph is strongly connected, there is a path between any two nodes u and w. Observe the path between a node u, $qual_u < Q$ and the closest node w for which $qual_w = Q$. Since we assume that $\forall v \in V$, $qual_v \leq Q$, then all the nodes on the path, and particularly the node just before w, have a *qual* variable $< Q$. In the next execution step of the node just before w, it will detect a better quality than its own, and will update its *qual* register to Q. Following that, its predecessors in the path will also update its their *qual* variable to Q. Eventually, so will u.

3. Follows directly from the code.

4. Follows directly from the code.

$\square$

The lemma shows that any fair execution of the protocol leads to configuration c, which is safe. Hence,

Corollary 11 *The BFS forest protocol for bidirectional general-topology networks is self-stabilizing.*