A Scalable Architecture for Reliable Distributed Multimedia Applications *

Fabio Panzieri and Marco Roccetti

Dipartimento di Matematica
Università di Bologna
Piazza Porta S. Donato 5
40127 Bologna (Italy)

Abstract

In this paper, we propose a communication software architecture designed to support distributed multimedia applications. We show that this architecture is scalable, and can provide high availability of the communication services.

1 Introduction

The purpose of this paper is to introduce the architecture of a *scalable* Distributed Computing and Communication Subsystem (DCCS), currently being designed in order to support reliable Distributed MultiMedia Applications (DMMAs). This architecture consists of algorithms, and protocols that implement those algorithms, for the structuring of the DMMAs, and for use from the DMMAs components for communication purposes. This architecture is scalable as it is designed to accommodate gracefully possible changes in size of the architecture itself.

We assume that a DCCS is constructed out of a communication infrastructure that supports the exchange of both isochronous, asynchronous, and synchronous information. That infrastructure can interconnect a great variety of nodes, characterized by diverse capabilities. In this context, by *scalable* DCCS we mean that the size of the DCCS itself can change, in terms of number of nodes and communication channels, according to specific user requirements. This notion of *scalability* entails that the performance and reliability of the algorithms and protocols that govern the DCCS components can at most degrade gracefully, as a consequence of modifications of the DCCS size; however, this possible degradation should be sufficiently contained not to jeopardize the DCCS services.

The class of DMMAs we consider include applications such as videotelephony, Computer Supported Cooperative Work (CSCW), and teleconferencing. These applications are characterized, principally, by the following requirement, which we term the IS requirement.

IS: independent streams of data, originated from possibly geographically dispersed and heterogeneous (i.e. continuous and discrete) input devices, are to be integrated so as to form a composite stream of synchronized data objects to be rendered (i.e. played out) at a collection of output devices.

Owing to the above IS requirement, it has been pointed out (e.g. [6, 1]) that DMMAs exhibit typical real-time requirements with respect to issues such as timeliness in data capturing, communication, and rendering. In order to meet these requirements, the DCCS architecture we propose embodies a collection of communication protocols, structured as a hierarchy of layers, that provide timely integration and synchronization of the DMMAs data streams. We term the service implemented by these protocols the "Synchronization Service".

In addition, the DMMAs may require, for reliability purposes, that the subsystem used to support them provide services that are both timely and highly available, under specified fault hypotheses. In any such subsystem, highly available services can be provided by introducing redundancy of those services. To this end, it is crucial that the algorithms and protocols, embodied in our DCCS architecture, allow the DMMA designer both to trade some of the performance he is provided with (quantified in terms of communication

*Partial support for this work was provided by the Commission of the European Communities under ESPRIT Programme Basic Research Project 6360 (BROADCAST), by the Italian National Research Council (CNR) under contract n. 93.01927.CT12.115.25585, and by the Italian MURST.

bandwidth usage), for augmented reliability (quantified in terms of redundancy of components), and to meet possible timeliness and high availability application requirements. Hence, those algorithms and protocols are to be scalable, according to the characterization of scalability introduced earlier.

This paper is structured as follows. In the next Section we discuss the principal issues we have addressed, and motivate our design decisions. In Section 3 we introduce the algorithms we propose in order to construct a scalable DCCS. In Section 4 we describe the structuring of the DCCS communication software we are designing. Finally, in Section 5, we provide some concluding remarks.

2 Design Issues

The principal issues that we have addressed in the design of our DCCS architecture include: i) the provision of what we have termed "scalability support", i.e. the complex of algorithms and protocols that allow our architecture to deal with modifications of the DCCS size (yet maintaining the expected DCCS services), ii) the choice of a particular data stream synchronization policy that meet the IS requirement, and iii) the definition of the fault model that our architecture is intended to cope with. Below, we discuss each of these issues in turn.

2.1 Scalability Support

As pointed out in [12], the principal motivation for the design of a scalable DCCS architecture is that scale is a primary factor that can influence the design and implementation of a distributed system. In particular, mechanisms that work adequately in small distributed systems may fail to do so when deployed within the context of larger systems. In view of this remark, one can observe that the complexity entailed by meeting the above IS requirement can be exacerbated by the need to provide scalability of the DCCS implementing the data stream synchronization policy.

The following three principal policies have been proposed in order to meet the IS requirement [9], termed *synchronization at the source*, *synchronization at the destination*, and *synchronization at the network*. A description of these policies is beyond the scope of this paper; hence, we shall not introduce them here. However, we would like to point out that each of these three policies shows several drawbacks; in particular, none of them scales well, as discussed in [11]. The approach we propose in order to meet the IS requirement is based on the use of a scalable, fault tolerant, decentralized synchronization policy, obtained as an extension of the algorithm described in [14]. This algorithm operates on a hierarchical tree structured virtual interconnection architecture, and manages synchronized integration of multimedia data streams (see Subsection 2.2). In [14], it is assumed that each source generates media packets at a constant rate, and that the communication delays are bounded in a time interval. Under these assumptions, the proposed algorithm minimizes the difference between the generation time of the data packets that are being synchronized, in the absence of globally synchronized clocks. By reducing the packet generation time differences, this algorithm minimizes the buffering time and space requirements of the data packets.

In contrast, our approach is founded on the following assumptions. Firstly, we place no conditions on the data packet generation rate. Secondly, we assume that bounded delay jitter is guaranteed by the communication subsystem; in particular, that subsystem can provide delay guarantees, sufficient buffer space, and a quite accurate clock synchronization by implementing, for example, the delay jitter control scheme described in [4].

The synchronization strategy that we propose operates on an abstraction of the physical communication infrastructure mentioned earlier. (Note that, in this paper, we are not concerned with the particular integration algorithm, e.g. [13], used to construct a composite multimedia data stream.) This abstraction consists of a robust, hierarchical, tree structured, virtual interconnection architecture. This architecture is structured as a complete M-ary tree with N leaves. In this tree, at most k spare links can be added to each node in order to obtain redundancy for tolerating a number of communication faults, which is dependent of k. We have termed this architecture *k-Augmented M-ary Tree* (k-AMT). The leaf nodes of the k-AMT represent the multimedia data sources and destinations of a given instance of a DMMA. Non-leaf nodes represent *synchronizers* of multimedia data streams, that implement the Synchronization Service mentioned in the previous Section. A link between two nodes represents a virtual communication channel between those nodes. Nodes connected by a virtual channel can communicate by exchanging messages.

To conclude this Subsection, we wish to point out that the principal motivation that has led us to investigate the design of the k-AMT hierarchical architecture is that, as shown in [14], hierarchical architectures can

scale beyond an order of magnitude than purely centralized or distributed architectures, while continuing to meet the application requirements.

2.2 Synchronization Policy

The synchronization policy proposed in [14] coordinates the activity of the nodes of the hierarchical tree-structured virtual interconnection architecture mentioned earlier. This policy operates as follows. Distinct leaf nodes can be clustered together, according to some physical or logical neighborhood criteria. Each cluster of leaf nodes may include "source" nodes that generate so-called "individual data streams" to be integrated in order to form a "composite multimedia data stream". In addition, each cluster may include "destination" nodes that perform the rendering of a composite multimedia data stream.

The parent node of a cluster of leaves receives data streams from its source leaves, integrates them to produce a composite multimedia data stream, and then transmits it to its immediate ancestor. This procedure is executed by each nonleaf node (i.e. the synchronizers) of the hierarchical virtual interconnection architecture, up to its root. The root node constructs the final multimedia data stream, and multicasts it to the destination leaves.

In our approach, we extend the synchronization policy introduced above in order to operate over a k-AMT. Our objective, in fact, is to combine both reliable communication and multimedia synchronization services. Thus, the k additional links at each node of the k-AMT, are used (see Subsection 3.3) for transmitting replicas of the individual data streams and of the composite multimedia data streams, in order to overcome problems that may arise from faults of both communication links and (nonleaf) nodes. We have chosen a complete and ordered M-ary tree (rather than an arbitrary rooted tree as in [14]), over which to implement the synchronization policy, in order to master and control the complexity of the algorithms and protocols we propose. In general, the abstraction of a MT structured interconnection architecture can be constructed out of a physical multimedia distributed system, as described in the following example.

Assume that a multimedia distributed system consist of a communication network that interconnects: (i) a workstation equipped with three input devices, e.g. a camera, a microphone, and a keyboard, and a video monitor output device, (ii) a workstation equipped with two output devices only, e.g. a video monitor and a loudspeaker, and (iii) a multimedia file server and a camera connected to the communication network via a local area network. In order to construct the required binary tree abstraction, the I/O devices can be uniquely identified by integer valued labels. The root node of that tree can be created in one of the workstations (and implemented by a specific synchronizer process), and each I/O device can be represented as a leaf node of that tree. Those devices can be clustered in the following four clusters, according to, for example, a physical neighbourhood criterion. A first cluster can consist of the camera and the file server, labeled 8 and 9, respectively. A second cluster can include the loudspeaker 10, and the video monitor 11; a third cluster can include the video monitor 12 and the keyboard 13. Finally, a fourth cluster can consist of the microphone 14, and the camera 15. (Note that, for the purposes of this example, we have assumed $M = 2$; an MT construction algorithm that works for arbitrary even values of M is proposed in Subsection 3.2.)

The activity of each of these clusters of devices is to be managed by a separate synchronizer process. Hence, in our example, four synchronizer processes are required. A synchronizer process can be represented as a node linked to the leaf nodes of the cluster that synchronizer is managing, and can be identified by a unique integer valued label. Yet again, owing to our initial assumption that $M = 2$, the activity of each pair of synchronizers is to be coordinated by a further synchronizer process. Thus, two such processes are required; each of these processes can be represented as a node in the tree we are constructing, linked to a pair of synchronizers, and to the root node. Figure 1 illustrates the nodes representing both the I/O devices (i.e. nodes 8 to 15) and the synchronizers (i.e. nodes 1 to 7) of our example. In Section 3 we show that the synchronization policy we propose scales well, and can tolerate a number of faulty links, which depends on k, at the cost of communication overhead, estimated in terms of number of messages. This cost can be quantified as proportional to N^2, (i.e. the square of the number of sources and destinations of multimedia data streams), and dependent both of k (i.e. the number of additional links per node) and of M (i. e. the number of sons of each node).

2.3 Fault Model

Faults have been classified as benign and byzantine faults (e.g. [10]). Benign faults include omission and timing faults; byzantine faults are those that exhibit an arbitrary or even malicious behavior. The fault model we consider consists of benign faults only, occurring at the virtual communication channels, and

at the non root (see below) nodes. Typically, these faults may cause message omissions, and delays. In this context, non-root nodes are characterized by fail-stop failure semantics, i. e. they either produce the expected result or no result at all. Thus, in the following, we assume that the fault of a node corresponds to a fault in the communication channels connecting that node to the k-AMT. In addition, we assume that these faults are "permanent", i.e. it is either impossible, or too expensive (e.g. in terms of additional communication delays), to recover from them within a particular instance of a DMMA.

Finally, faults at the leaf nodes fall outside the scope of our fault model (as these nodes are not concerned with the implementation of the synchronization service). Instead, faults at the root node can be dealt with either by using conventional replication techniques, or by extending the algorithm proposed in the next Section to incorporate dynamic reconfiguration strategies (this possible extension is not discussed here).

3 Scalability Support

In this Section we first introduce the definition of the k-AMT. Next, we describe two algorithms that manage the k-AMT, namely the *k-AMT Construction* (k-AMTC) and the *k-AMT Synchronization* (k-AMTS) algorithms. For the purposes of this discussion, we shall assume that no faults occur during the execution of the k-AMTC algorithm; instead, faults can occur during the execution of the k-AMTS algorithm.

3.1 k-AMT Definition

Let MT be a complete M-ary tree with N leaf nodes, where M is an even integer, and $N = M^{d-1}$ (d being the depth of the tree). The total number of nodes in MT is $T = (M \times N - 1)/(M - 1)$.

Each node in MT can be uniquely identified by an integer valued label i, such that $1 \leq i \leq T$, termed "node identification number". The root node is assigned label $i = 1$; the M children of each node i can be labeled from left to right with the sequence of consecutive integers generated by $M \times i - (M - 2) + j$, $j = 0, 1, \ldots, (M - 1)$. Each node i in MT resides at an MT level $l(i) = \lfloor \log_M i \rfloor$, $1 \leq i \leq T$; hence, the root is at MT level 0, the leaves are at MT level $d - 1$.

Note that, given a node i such that $l(i) < d - 1$, the identification numbers of the leaves of the subtree rooted at i can be produced by the sequence of consecutive integers generated by

$$M^{(d-l(i)-1)} \times i - (M - 2) \times \sum_{k=0}^{d-l(i)-2} M^k + j$$

where $j = 0, 1, \ldots, M^{(d-l(i)-1)} - 1$.

An MT defined as above, and characterized by a depth $d > 2$, can be augmented so as to form a k-AMT by adding at most k redundant links to each of its nodes, with $1 \leq k < d-1$. In particular, given the value k, any node i can be linked to those nodes at the same level l as i itself, whose labels are $j = i + M^h$, with $1 \leq h \leq k$, $\lfloor (i+p)/M^h \rfloor$ even and greater than zero, and $p = (M^l - ((M^l - 1)/(M - 1) + 1))$. Those nodes will share with i a common ancestor at the level $l - h - 1$ (with $1 \leq h \leq k$, and $h < l$).

For example, the MT mentioned in Subsection 2.2 is characterized by $M = 2$, $d = 4$, $N = 8$, $T = 15$. The k-AMT of Figure 1 can be obtained from that MT by introducing a $k = 1$ redundancy of the MT links; in particular, the redundant links are those connecting the pairs of nodes (4,6) and (5,7) at level 2, and (8,10), (9,11), (12,14), and (13,15) at level 3. As shown in Figure 1, the nodes at the end points of these links share a common ancestor other than the parent node.

3.2 k-AMTC Algorithm

At start up time, the k-AMTC algorithm is provided with a description of the DMMA as consisting of a collection of *sites*. Each site is identified by a unique *site address* that can be used in order to communicate with that site. A site is elected to be the root node of the k-AMT (the root election algorithm is beyond the scope of this paper). The k-AMTC algorithm operates as follows; the root node executes the following three steps.

1. The root receives as input the values of k, M, d, and of the addresses of the N sites that represent the sources and the destinations of the DMMA data streams. It sets its identification number to 1, and its level $l(1) = 0$. If the M and d input values are such that no complete M-ary tree with N leaves can be constructed, the root introduces a sufficient number of additional sites, termed "virtual leaves", so as to form the required complete M-ary tree. These virtual leaves will act only as data stream repeaters, i.e. they will not act as data sources or destinations. (In the following, N indicates the total number of leaves of the k-AMT, including possible virtual leaves.)

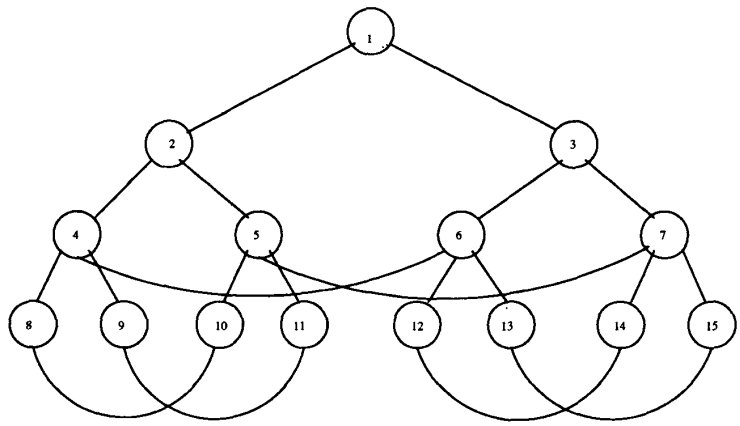

Figure 1: k-AMT

2. The root assigns to the N leaves their corresponding identification numbers (calculated as described in the previous Subsection); then, it groups the N leaves into $M^{(d-2)}$ clusters consisting of M leaves each.

3. The root elects M distinct sites, that are not leaves, to be its own son nodes. Then, it creates a link with each son i, and transmits to it the following parameters: its own identification number, the identification number i, the level value associated to i (i.e. $l(i) = 1$), the values of M, d, and k, and, finally, the set of $M^{d-l(i)}$ identification numbers, together with their corresponding addresses, of the leaves belonging to the subtree rooted at i (calculated as introduced in the previous Subsection).

Each node i, other than the k-AMT root, executes the following two steps.

4. Node i maintains both its own identification number i, its parent identification number, the value of its own level $l(i)$, and the values of M, d, and k. It calculates and maintains the identification numbers of the nodes with which it is to be connected by means of the k spare links, and establishes the appropriate links with those nodes.

5. Node i elects M distinct unlabelled sites to be its own sons. It creates a link with each son j and provides it with: its own identification number, the identification number j, the level value associated to j (i.e. $l(j) = l(i)+1$), the values of M, d, and k, and finally the set of $M^{d-l(j)-1}$ identification numbers, together with their corresponding addresses, of the leaves belonging to the subtree rooted at j.

As mentioned above, steps 4 and 5 are identically executed by each node i, such that $0 < l(i) < d-1$, with the exception of the nodes at level $d-2$. In fact, each node j at that level does not have to select its own son nodes, in step 5 above, as j has been already provided with the addresses and identification numbers of those nodes by its parent node. Finally, each node, whose level is equal to $d-1$ (i.e. a leaf node of the k-AMT), executes only step 4 of the above algorithm.

To conclude this Subsection, the following three observations are in order: i) The k-AMTC algorithm can be executed, in a distributed fashion, in $O(\log_M N)$ time; ii) for the scope of this paper, we have not investigated fault tolerance policies to be incorporated in the k-AMTC algorithm; iii) we have assumed that the values of M and d are provided as input to the root. (These two input values can be replaced by appropriate parameters specifying, for example, particular Quality Of Service (QOS) [8] values that allow the root node to calculate M and d).

3.3 k-AMTS Algorithm

The k-AMTS algorithm implements the synchronization strategy introduced in Subsection 2.2. This algorithm operates over a k-AMT in two distinct phases, as described below. In the first phase (termed *Collect Phase*), the parent node of a cluster of leaves receives data streams from its (source) leaves, and integrates them to produce a composite multimedia data stream. The data object constructed to represent that

composite multimedia data stream includes the identification numbers of those k-AMT leaf nodes that originated the individual data streams used to construct that composite stream. Finally, that node transmits to its immediate ancestor the composite stream it has produced.

This procedure is executed by each nonleaf node of a k-AMT, up to the root of the k-AMT. The root node constructs the final synchronized multimedia data stream and then, in the second phase (termed *Disseminate Phase*) "multicasts" it to the (destination) leaves. The *Collect* and *Disseminate* phases are described below in detail.

Collect Phase In this phase, each nonleaf node i executes the integration of the data streams originated from its own son nodes, if the set of identification numbers provided with those streams matches the set of the identification numbers which identify the leaves of the subtree rooted at i. In this case, the data stream integration executed by node i is said to produce a composite multimedia data stream for the node i. A composite multimedia data stream for the root node is termed the final synchronized multimedia data stream.

In case a mismatch occurs between the set of stream identification numbers received at i and the set of identification numbers of the leaves rooted at i, the *Collect Phase* uses the additional links at each node in order to overcome problems that may arise from faulty communication links, as described in the following.

- Each (source) k-AMT leaf node i sends to its immediate ancestor the data stream it originates, together with its own identification number. In addition, it sends "replicas" of that stream (and of its own identifier) to each node linked with i through a spare link.
- Each nonleaf node operates in three distinct sequential sub phases: the reception sub phase, the synchronization sub phase, and the transmission sub phase. For the sake of clearness, we will introduce these three sub phases in a non sequential order, and itemize the principal actions to be carried out in each sub phase:

  Transmission sub phase,

  - Each nonleaf node i sends to its immediate ancestor its composite multimedia data stream (if any, see below), whose integration it has just completed.
  - Each nonleaf node i forwards to its immediate ancestor, and to each node j linked with i through an additional link, the "first instance" of each received replica. Note that, as in a k-AMT multiple paths exist between any two nodes, a data stream originated from a source node j may reach a synchronizer node i more than once (hence the use of the phrase "first instance" above); in addition, i does not send to j replicas it has received from j itself.

  Reception sub phase

  - Each nonleaf node i receives from its M sons both the forwarded replicas and the composite multimedia data streams they transmit. If the node i fails in receiving the composite multimedia data streams from any of its M son nodes, either because a fault has occurred, or because those nodes have not produced it, then node i raises an exception and discards all the composite multimedia data streams received from all the other sons (the motivation for this operation is discussed later).
  - Each nonleaf node i receives from each node j, linked with i through a spare link, all the replicas transmitted by j.

  Synchronization sub phase

  - In the case in which no exception has been raised, node i executes the integration of the composite multimedia data streams it has received from its sons. The resulting multimedia data stream will be a composite multimedia data stream for node i. Instead, if an exception has been raised, the node i examines all the replicas it has received. If a set of the identification numbers, attached to some replicas, exists which matches the set of the identification numbers identifying all the leaves of the subtree rooted at i, then the integration of these replicas is executed. The resulting multimedia data stream constitutes a composite multimedia data stream for the node i. Otherwise, no composite multimedia data stream for node i can be produced; thus, the node i will act as replica repeater only.

Disseminate Phase In this phase, the root of the k-AMT transmits a copy of the final synchronized multimedia data stream, whose synchronized integration

it has just executed, to each son. This procedure is executed, by each nonleaf node of the k-AMT, which transmits copies of the final synchronized multimedia data stream to all its sons, down to the leaf destination nodes. The motivation for this operation is to use the additional links at each node in order to overcome problems that may arise from faulty links, as described below.

- Each nonleaf node i, upon receiving the first copy of the final synchronized multimedia data stream, forwards it both to each son and to each node linked with i through a spare link.
- Each (destination) leaf i, upon receiving the first copy of the final synchronized multimedia data stream, forwards it to each leaf node, linked with i through a spare link, and then plays it out.
- Each leaf node i, which is not a destination, forwards to each leaf node the first copy of the final synchronized multimedia data streams it receives through its spare links.

The following definitions are introduced in order to characterize the successful execution of the k-AMTS algorithm.

Definition 1. A *Collect Phase* over a k-AMT terminates successfully *iff* it produces the final synchronized multimedia data stream at the root of the k-AMT. □

Definition 2. A *Disseminate Phase* over a k-AMT terminates successfully *iff* the corresponding *Collect Phase* has terminated successfully, and the final synchronized multimedia data stream, produced by that *Collect Phase*, has reached all the destination leaves. □

Definition 3. A k-AMTS algorithm terminates successfully *iff* its *Disseminate Phase* terminates successfully. □

The following theorems and lemmas are presented in order to provide sufficient conditions for a successful termination of the k-AMTS algorithm. (Formal proofs of these theorems and lemmas can be found in [11]).

Theorem 1. Let r_l be the set of links between the nodes at level $l-1$ and the nodes at level l. Let s_l be the set of the spare links (if any) between the nodes at level l. A *Collect Phase* terminates successfully if at most $S' = (2 \times k \times (d-k-1) + k \times (k+1))/2$ link failures occur, and out of S':

1. no more than k faults occur at $r_l \cup s_l$, for each l such that $k < l \leq d-1$,
2. no more than $l-1$ faults occur at $r_l \cup s_l$, for each level l such that $0 < l \leq k$. □

A set of at most S' faults, occurring during a *Collect Phase* and satisfying the conditions of Theorem 1, is said to be *innocuous*.

Lemma 1. A *Collect Phase* terminates successfully, for any acceptable value of k, if at most 1 link failure occurs. □

Theorem 2. Let r_l and s_l be defined as in Theorem 1. A *Disseminate Phase* terminates successfully if at most $S'' = (2 \times k \times (d-k-1) + k \times (k+1))/2$ link failures occur, and out of S'':

1. no more than k faults occur at $r_l \cup s_l$, for each l such that $k < l \leq d-1$,
2. no more than $l-1$ faults occur at $r_l \cup s_l$, for each level l such that $0 < l \leq k$. □

A set of at most S'' faults, occurring during a *Disseminate Phase* and satisfying the conditions of Theorem 2, is said to be *innocuous*.

Lemma 2. A *Disseminate Phase* terminates successfully, for any acceptable value of k, if at most 1 link failure occurs. □

Corollary 1. A k-AMTS algorithm terminates successfully if an *innocuous* set of faults occurs during its *Collect Phase* and an *innocuous* set of faults occurs during its *Disseminate Phase*. □

The following Propositions and Corollaries show that the proposed policy scales well, and tolerates a number of faulty links at the cost of message overhead. (Formal proofs are available in [11]).

Proposition 1. A number of messages $Z' < (M \times N - 1/(M-1)) \times (k+1) \times (N+1)$ is necessary to execute successfully a *Collect Phase* over a k-AMT with N leaves. □

Proposition 2. A number of messages $Z'' < ((M \times N - 1)/(M-1)) \times (M+k)$ is necessary to execute successfully a *Disseminate Phase* over a k-AMT with N leaves. □

Corollary 2. A number of messages $Z = O(N^2)$ is necessary to execute a k-AMTS algorithm over a k-AMT with N leaves. □

In conclusion, the k-AMTS algorithm message overhead, estimated in terms of number of messages, results to be proportional to N^2 (i.e. the number of sources and destinations of multimedia data streams), and dependent of both k (i.e. the number of additional links per node), and M (the number of sons for each nonleaf node in the k-AMT).

This result can be compared to that obtained from a fully decentralized synchronization policy, and that obtained from a centralized synchronization policy. With the former policy, the message overhead is (in

the worst case) proportional to N^3, if we assume that this policy is deployed in a fully interconnected network architecture by implementing N atomic reliable broadcasts [7]. However, this policy can tolerate up to $N-2$ link failures. In contrast, with the latter policy, the message overhead is proportional to N, but it tolerates no link failures. In view of this observation, we claim that our architecture is highly scalable with respect to bandwidth consumption.

4 Communication Protocols

In general, the orchestration and coordination activities to be performed by a DMMA require communication support that allows the implementation of the following three activities: i) synchronized playing out at a single destination node of multimedia data originated from multiple, possibly distributed, source nodes; ii) synchronized playing out, at a collection of destination nodes, of integrated multimedia data originated from a single source node; iii) synchronized playing out, at a collection of destination nodes, of integrated multimedia data originated from multiple (yet again, possibly distributed) source nodes.

Thus, the principal communication models required by a DMMA are those for *many to one*, *one to many* and *many to many* communications. Hence, the DCCS communication software architecture that we propose consists of a collection of protocols that support reliable group communications, over the above defined k-AMT abstraction. In particular, communications on the k-AMT are implemented by message diffusion protocols that allow their users to transmits messages to multiple destinations. Those messages are effectively transmitted to their destinations via different routes, transparently to the protocol users. The protocols ensure that, under predefined failure hypotheses, the transmitted messages reach their destination.

The k-AMT communication software can be thought of as hierarchically structured in two principal layers (or levels of abstraction); the basic communication interface supporting that architecture is assumed to provide a Real-Time Transport Service (RTTS) such as that described in [15], [3]. This service provides real-time communications, based on the management (i.e. establishment, maintenance, and release) of a so-called *real-time virtual channel*, between two single nodes in the k-AMT. A real-time channel guarantees an upper bound of the delays that can be experienced by the packets on the channel. The sending site is notified with a positive (or negative) acknowledgement if the real time channel has been established (or not established).

The structuring of the architecture we propose can be summarized as follows. The lower level (*Level 1*) incorporates a so-called *Multicast Virtual Circuit Protocol* (MVCP). This protocol uses the supporting RTTS to provide timely multicasting of real-time data streams over a collection of real-time virtual channels [2]. The higher level (*Level 2*), instead, exploits the MVCP services to implement the following two protocols: the *Augmented Tree Protocol* (ATP), that constructs the k-AMT characterized by redundant links, introduced earlier, and the *2 Phase Mixed Media Protocol* (2PMMP) that provides communications over the k-AMT. We introduce below the above mentioned protocols.

Multicast Virtual Circuit Protocol (MVCP) This protocol provides timely multicasting of multimedia data streams over a collection of real-time virtual channels. In particular, MVCP manages a collection of h real-time virtual channels between a sending node and h different receiving nodes, of a k-AMT. The principal performance requirement to be met in order to establish each of the h real-time channels, is that each channel guarantee the same delay bound D.

The sending node is notified (by means of the above mentioned acknowledgments provided by the real-time service) if any, say w, of the h real-time channels cannot be established under the imposed condition of a delay bound equal to D. Nevertheless the successfully established $h-w$ real-time channels are used to transmit multimedia data streams. If the number w of not established channels is greater than 0, the MVCP returns an exception, termed MVCP *type 1* exception, which indicates the number w of not established channels.

After a real-time channel has been successfully established between two nodes, a failure of that channel can damage the functionality of the real-time communication service. In order to overcome this problem, the receiving node of each real-time channel sends a positive acknowledgment back to the sending node only if the expected multimedia data stream has reached the receiving node within the guaranteed predefined delay. If the sending node receives a number of acknowledgments less than h, then an exception (termed MVCP *type 2* exception) is raised, which reports the number v of not received acknowledgments.

Each receiving node maintains the number t of acknowledgments it has transmitted.

Augmented Tree Protocol (ATP) This protocol constructs the k-AMT abstraction by implementing

the k-AMTC algorithm, described earlier.

The implementation of that algorithm can be summarized as follows. After each node i has identified its M son nodes, and the k peer nodes to which it is to be connected through the spare links, it establishes (using the MVCP interface primitives) two separate multicast virtual circuits, termed $MVC_1(i)$ and $MVC_2(i)$. $MVC_1(i)$ connects node i with its son and peer nodes; thus, the number of real time virtual channels that form $MVC_1(i)$ is $h = M + k$, if $l(i) > k$, or $h = M + l - 1$ otherwise. Instead, $MVC_2(i)$ connects node i with its parent and peer nodes; hence, the number of real time virtual channels that form $MVC_2(i)$ is $h = k + 1$, if $l(i) > k$, and $h = l$ otherwise.

In addition, each time a new source or destination site requires to join an existing k-AMT, firstly the ATP searches for an existing *virtual* leaf node of that k-AMT. If a *virtual* node exists, ATP assigns that node to the new site. Otherwise, the k-AMTC algorithm is to be executed to construct a new k-AMT. Instead, if a source or destination node wishes to leave an existing k-AMT, that node (depending on specific application requirements) can either be considered as a *virtual* node of the k-AMT, or the k- AMTC algorithm can be executed to construct a new k-AMT that does not include that node.

The termination of the ATP entails that each node in the k-AMT is ready to execute the *Collect Phase* of the k-AMTS algorithm (see below). However, it is worth observing that, during the execution of the ATP, possible MVCP *type 1* exceptions can be raised. In general, ATP intercepts those exceptions, and terminates indicating that the construction of the required k-AMT has failed. In particular, if at least one of these MVCP exceptions is such that $h - w = 0$, the sufficient condition of Theorem 1 has been violated; hence, ATP terminates with a failure exception indicating that a node is isolated. Otherwise, it terminates with a failure exception that indicates that the k-AMT construction has failed; however, at least one communication path between each pair of nodes exists. Typically, the handling of the ATP exceptions is an application specific issue; as we have assumed that the k-AMTC algorithm always terminates successfully, we will not discuss any specific strategy for handling those exceptions in this paper.

2 Phase Mixed Media Protocol (2PMMP) This protocol supports the communications over a k-AMT by implementing the k-AMTS algorithm described in Subsection 3.3.

The protocol can be thought of as implemented by two primitive operations, named *collect* and *disseminate* respectively, that allow each k-AMT node to participate to the *Collect Phase* and the *Disseminate Phase* of the k-AMTS algorithm.

The *collect* primitive is invoked at the root node of the k-AMT in order to start the *Collect Phase* of the k-AMTS algorithm. The implementation of this primitive causes that each k-AMT node i invokes the MVCP interface primitives in order both to read from its son and peer nodes via $MVC_1(i)$, and to write to its parent and peer nodes via $MVC_2(i)$.

The termination of the *collect* primitive returns a *partial collect* exception if at least one MVCP *type 2* exception has been raised, with $(h - w) - v = 0$. (In this case, in fact, the sufficient conditions of Theorem 1 have been violated).

However, whether or not a *partial collect* exception has been raised, the execution of the *collect* primitive cannot be said to terminate successfully until the root node has constructed the final synchronized multimedia data stream (as from Definition 1 in Subsection 3.3). Thus, a *final collect* exception is raised, only if the root of the k-AMT is not able to compose the final synchronized multimedia data stream.

The *disseminate* primitive is invoked at the root node of the k-AMT in order to transmit the final multimedia data stream to the destination leaf nodes of the k-AMT. The implementation of this primitive causes that each k-AMT node i invokes the MVCP interface primitives in order both to write to its son and peer nodes via $MVC_1(i)$, and to read from its parent and peer nodes via $MVC_2(i)$.

The termination of the *disseminate* primitive returns a *partial disseminate* exception if at least one k-AMT node has recorded a number $t = 0$ of acknowledgments it has sent. (In this case, in fact, the sufficient conditions of Theorem 2 have been violated). However, a *final disseminate* exception is raised only if at least one destination node has not received the final synchronized multimedia data stream (as from Definition 2 in Subsection 3.3). In conclusion, if no *final disseminate* exception has been raised, 2PMMP terminates successfully (as from Corollary 1 of the previous Subsection).

5 Concluding Remarks

In this paper, we have introduced the architecture of a scalable DCCS designed to support reliable DMMAs. We claim that the DCCS architecture we propose can support highly available services and assist in the implementation of scalable distributed multimedia

applications. We believe that the experimental evaluation of the protocols and algorithms we have described can lead us to confirm or modify the design concepts we have adopted. To this end, a prototype implementation of the proposed DCCS architecture is currently being developed at our Department, based on the *x*-Kernel [5]. An extended version of this paper, referenced as [11] in the bibliography below, is available via anonymous FTP from the area `ftp.cs.unibo.it:/pub/TR/UBLCS`.

Acknowledgements We wish to thank our colleagues Lorenzo Alvisi (Cornell University), Özalp Babaoğlu and Lorenzo Donatiello (Università di Bologna), Santosh Shrivastava (University of Newcastle upon Tyne), and the anonymous referees of the Fourth International Workshop on Network and Operating System Support for Digital Audio and Video, for their useful comments on an earlier version of this paper.

References

[1] G. Coulson, F. Garcia, D. Hutchison, D. Shepherd, *Protocol Support for Distributed Multimedia Applications*, in **Network and Operating System Support for Digital Audio and Video**, R.G. Herrtwich (Ed.), LNCS 614, Springer-Verlag, Berlin Heidelberg, 1992, pp. 45 - 56.

[2] D. Ferrari, D. C. Verma, *A Scheme for Real-Time Channel Establishment in Wide Area Networks*, IEEE JSAC, SAC-8, April 1990, pp. 368 - 379.

[3] D. Ferrari, A. Gupta, M. Moran, B. Wolfinger, *A Continuous Media Communication Service and its Implementation*, Proc. Globecom'92, Orlando, Florida.

[4] D. Ferrari, *Design and Application of a Delay Jitter Control Scheme for Packet-switching Internetworks*, in **Network and Operating System Support for Digital Audio and Video**, R.G. Herrtwich (Ed.), LNCS 614, Springer-Verlag, Berlin Heidelberg, 1992, pp. 72 - 83.

[5] N. C. Hutchinson, L. L. Peterson, *The x-Kernel: An Architecture for Implementing Network Protocols*, IEEE Trans. on Software Engineering, Vol. SE 17, N. 1, January 1991, pp. 64 - 76.

[6] K. Jeffay, D. L. Stone, F. Donelson Smith, *Kernel Support for Digital Audio and Video*, in **Network and Operating System Support for Digital Audio and Video**, R.G. Herrtwich (Ed.), LNCS 614, Springer-Verlag, Berlin Heidelberg, 1992, pp. 10 - 21.

[7] T. A. Joseph, K. P. Birman, *Reliable Broadcast Protocols*, in **An Advance Course on Distributed Systems**, S. J. Mullender (Ed.), Addison Wesley Publishing Co., 1989, pp. 313 - 338.

[8] J. Kurose, *Open Issues and Challenges in Providing Quality of Service Guarantees in High Speed Networks*, ACM SIGCOMM Computer Communication Review, Vol. 23, N. 1, January 1993, pp. 6 - 15.

[9] P. Leydekkers, B. Teunissen, *Synchronization of Multimedia Data Streams in Open Distributed Environments*, in **Network and Operating System Support for Digital Audio and Video**, R.G. Herrtwich (Ed.), LNCS 614, Springer-Verlag, Berlin Heidelberg, 1992, pp. 94 - 104.

[10] P. M. Melliar-Smith, L. E. Moser, V. Agrawala, *Broadcast Protocols for Distributed Systems*, IEEE Trans. on Parallel and Distributed Systems, Vol. 1, N. 1, January 1990, pp. 17 - 25.

[11] F. Panzieri, M. Roccetti, *A Scalable Architecture for Reliable Distributed Multimedia Applications*, Tech. Rep. UBLCS TR 93-23, Università di Bologna, Bologna (Italy), October 1993.

[12] M. Satyanarayanan, *The Influence of Scale on Distributed File System Design*, IEEE Trans. on Software Engineering, Vol. 18, N. 1, January 1992, pp. 1 - 8.

[13] D. Sheperd, M. Salmony, *Extending OSI to Support Synchronization Required by Multimedia Applications*, Computer Communications, Vol. 13, N. 7, September 1990, pp. 399 - 406.

[14] P. Venkat Rangan, H. M. Vin, S. Ramanathan, *Communication Architectures and Algorithms for Media Mixing in Multimedia Conferences*, IEEE/ACM Trans. on Networking, Vol. 1, N. 1, February 1993, pp. 20 - 30.

[15] B. Wolfinger, M. Moran, *A Continuous Media Data Transport Service and Protocol for Real-time Communication in High Speed Networks*, in **Network and Operating System Support for Digital Audio and Video**, R.G. Herrtwich (Ed.), LNCS 614, Springer-Verlag, Berlin Heidelberg, 1992, pp. 171 - 182.