UNIVERSITÀ DEGLI STUDI DI ROMA "LA SAPIENZA"

DOTTORATO DI RICERCA IN INFORMATICA

XV CICLO – 2005– XV-05-2

Reliable Secure Multicast Data Transmission and Applications

Flavio Lombardi

UNIVERSITÀ DEGLI STUDI DI ROMA "LA SAPIENZA"

DOTTORATO DI RICERCA IN INFORMATICA

XV CICLO - 2005– XV-05-2

Flavio Lombardi

Reliable Secure Multicast Data Transmission and Applications

Thesis Committee

Prof. Giancarlo Bongiovanni (Advisor)
Prof. Luigi Vincenzo Mancini
Ing. Mauro Draoli

Reviewers

Prof. Domenico Laforenza
Prof. Peter Kunszt

AUTHOR'S ADDRESS:

Flavio Lombardi
Dipartimento di Informatica
Università degli Studi di Roma "La Sapienza"
Via Salaria 113, I-00198 Roma, Italy
E-MAIL: lombardi@di.uniroma1.it
WWW:

Abstract

Multicast transmission is a useful technology for one to many and many to many communication scenarios. It allows to interchange data among members of a group in a bandwidth-efficient way.

During its experimental phase IP multicast has proven to be a technology more scalable than unicast, but nevertheless limited, concerning reliability, security and scalability aspects in the real world. Solutions proposed in literature are not complete and have not had a wide acceptance by the research and the engineering communities. The reason why is that many complex real-world applications that would benefit from multicast transmission require a standard, widely adoptable, mature and efficient protocol set that does not exist yet.

This thesis collects research investigations about the main topics in multicast transmission: performance, security and applications. In our aim to study, improve and validate reliable multicast protocol solutions, we have created a software framework allowing for multicast protocol implementation, monitoring and performance measurement. Such a Java open source framework contains building blocks that can be extended and combined in order to implement the desired protocol features and collect experimental data.

Java has been adopted as a reference platform by our research group. It guarantees portability, particularly interesting today thanks to the mass-market availability of mobile and set-top devices supporting it. This allows us to investigate the potentiality of such a widely adopted technology in the design and implementation of largely scalable multicast protocols.

The first module of our framework is JMFTP, an implementation of a scalable reliable multicast bulk data transfer protocol. We analyze the throughput and robustness of JMFTP and individuate trade-offs among protocol parameters. JMFTP proves extremely robust with respect to data loss and delay, rendering it particularly suitable for satellite networks.

A further module of our framework is the implementation of a secure key exchange protocol named MTLS, providing secure group membership management and session key data encryption.

We integrate cryptography and key exchange with JMFTP thus obtaining a secure reliable multicast transport protocol that can transfer bulk data from one sender to

multiple receivers throughout the Internet. Performance tests allow us to evaluate the impact of security on performance.

We then adopt the above framework to support data replica synchronization in a Grid computing infrastructure. Test results show that such a Grid application may benefit from the adoption of the proposed replication model while further research is being done on the subject.

Acknowledgements

Writing a Ph.D. Thesis is a large task that can be done only thanks to many people, who give their support both in scientific and personal matters.

I am glad to take this opportunity to mention, in the following list (probably incomplete because of the Ack-explosion-limiting algorithm I adopted in this scenario), all those people I owe a lot.

First of all, I would like to thank Dr. Carlo Gaibisso and Ing. Mauro Draoli. Together with them, I wrote my first research papers during the year 2000, they recognized in me some skill in doing research and greatly encouraged me when I was in trouble. Dr. Gaibisso gave me the chance to collaborate to the NetLab effort at Iasi since 1999.

I enjoyed a lot working with Prof. Giorgio Gambosi. I also owe a lot to Luca Becchetti and Aldo Stentella Liberati, who helped me writing my first papers.

A person deserves a sincere acknowledgement: Prof. Giancarlo Bongiovanni. As my thesis advisor, Giancarlo introduced me to the issues of computer networks and gave me the motivation to start a Ph.D. program. He followed my scientific work during the whole time of my Ph.D.

A special thank and a kiss goes to Sonia Simonetti.

Special thanks and kisses also go to my father Antonio, my mother Paola, my grandmother Giovanna, my brother Fabio, my two wonderful nephews Arianna and Federico and their mother Stefania.

I am grateful to Marco Bianchi@NetlabIasi and Roberto Puccinelli@Cnr, working with them is always a pleasure, even late at night.

Last, but not least, I would like to mention those who shared the Ph.D. years with me, among others Sonia ”Sophinia” Campa@Unipi, Edgardo Ambrosi@Unifi, Francesco ”Nagiosman” Ruffino@Iasi, Roberto Di Pietro, Antonio Durante, Fabio, Claudio, Giorgio @Dsi, All LipariSchool People and many many more...

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Multicast transmission is a useful technology for one to many and many to many communication scenarios. It can be used, for instance, for videoconferencing or for disseminating data at remote sites. The efficient data distribution to a set of target hosts geographically distributed is a well-known problem for telecommunication and computer networks.

During its experimental phase IP multicast has proven to be a technology more scalable than unicast, but nevertheless limited, concerning reliability, security and scalability aspects in the real world.

Solutions proposed in literature are not complete and have not had a wide acceptance by the research and the engineering communities. The reason why is that many complex real applications that would benefit from multicast transmission technology require a standard, widely adoptable, mature and efficient protocol set that does not exist yet.

1.1 Goals of the Thesis

In an aim to study, improve and validate reliable multicast protocol solutions, we collect in this thesis research investigations about some of the main topics in multicast transmission: performance, security and applications.

With the objective of describing the state of the art in these three areas, we start by examining present research achievements, briefly reported in this document.

In order to investigate the potentiality of such a widely adopted technology as Java in the design and implementation of a scalable multicast transport protocol, we have implemented a Java software framework allowing for modular multicast protocol implementation, monitoring and performance measurement.

An important goal of this thesis is the evaluation of the impact of security on the performance of a multicast data transmission. In order to do so, a secure key exchange protocol named MTLS has been created and performance tests have been

run on a variety of scenarios.

With the aim of studying the usefulness of multicast in other research areas, we apply our framework to a data replica management problem that can potentially benefit from our results.

In order to obtain experimental results for real-world applications, we adopt the above-mentioned framework to allow resources in a Grid computing infrastructure to efficiently update distributed copies of data.

1.2 Thesis Organization

This thesis is divided in three main sections:

1. Reliable Multicast;
2. Secure Multicast;
3. Reliable Multicast Applications.

This organization reflects three main areas of research I focused on during the years of my Ph.D. In the first section the main problems of reliable multicast are presented, along with a significative subset of solutions proposed up to date and the main contribution we have given. In the second section a glimpse of the complex scenarios of secure multicast is given, together with a view of the most interesting proposals to date. Our experience with multicast security is described. Last but not least in the third section a specific problem is analyzed such as the replica consistency management. A brief survey of present approaches and our contribution in the context of data Grids (focused on the European DataGrid) are presented. Finally a section is devoted to a problem we address (UDDI and Web Service instance selection) to which we think the multicast model can be applied with promising results.

More in detail the document is organized as follows: next chapter describes multicast routing and reliable multicast issues; chapter 3 contains a survey of the state of the art solutions for reliable multicast; chapter 4 explains the main contribution we have given in this area; chapter 5 introduces security issues; chapter 6 reviews the state of the art of secure multicast solutions; chapter 7 describes the main contribution we have given to the multicast security area; in chapter 8 the replica management problem is analyzed; chapter 9 illustrates the solutions we propose regarding the replica management problem in the Grid and Web Services and UDDI issues; finally, chapter 10 contains conclusion and further work.

Part I

Reliable Multicast

Chapter 2

IP Multicast

Multicast IP transmission enables the distribution of data packets to a group of receivers in an efficient way. It is described in RFC 1112[Dee89].

IP multicast main **features** are:

- It is based on UDP: datagrams are sent to a shared class D IP address to which all group members(or hosts) are listening (are joined).
- Group membership is open: hosts may attach to the group by joining or detach from the group by leaving.
- Group membership is dynamic: hosts can join and leave a multicast group at any time.
- Router support: IP routers forward datagrams to each of their downstream links that takes to hosts belonging to the multicast group. The IGMP protocol [CDF02] provides a mechanism for hosts to communicate with routers in order to join and leave multicast groups.

Main IP Multicast **issues** are:

- Congestion Control: the lack of congestion control mechanisms in multicast transport can cause inefficiencies regarding network bandwidth utilization. This is really dangerous since a single multicast group can affect large portions of a network. Other traffic, such as well-behaved TCP flows, may be penalized by poorly behaved multicast.
- Address Allocation: IP Multicast identifies a range of addresses (224.0.0.0 to 239.255.255.255) used to identify a multicast group. Collisions are possible where two multicast groups are established independently with the same multicast address assigned to both, possibly causing data intended for one group to be received by members of another group.

- TTL Scoping: containing transmission of multicast traffic to a subset of a network is often useful. The Time To Live (TTL) field in the header of IP packets limits the number of routers a packet can traverse. Some reliable multicast transport protocols adjust the TTL field of repair packets to limit their scope. However, using TTL to contain retransmission of packets does not always confine the data to the intended receivers.

- Scalability of Router Tables: as multicast is increasingly deployed, more router table space is consumed. When a large number of concurrent multicast groups are active, the exhaustion of table memory space or processing power in backbone routers is possible, and this is a scalability limitation.

A list of **requirements** varying for heterogeneous multicast applications are:

- Reliability: most applications require full reliability, whereas some (such as streaming video) can trade-off a certain amount of data loss in order to achieve high throughput and timely delivery.

- Real-time Performance: some applications require that data be delivered within a certain period of time. Others simply require that it be eventually delivered.

- Support for multiple senders: protocols allowing more than one sender at the same time can be more complex than those allowing only one.

- Late Joins: receivers that join the group after some data has been sent are known as late joins. If all receivers are required to have a single start time, protocol design is much simpler since late joins are not allowed. If receivers may start at any time and must receive copies of all data sent (late join with full data recovery), all data must be preserved. If receivers may start at any time but do not receive copies of all data sent earlier (late join with limited data recovery), transmitted data can be disposed of more easily.

- Ordering: most applications require data to be delivered in order, others do not.

- Consistency: some applications require that all data be delivered to all receivers at exactly the same time. Others have no such requirement.

A rough **classification** of multicast applications can be as follows:

- Bulk data applications: concerned with transferring files. Some examples are distribution of software updates and corporate data. Such applications typically require one sender per channel, no late joins, no real-time requirement, file level consistency, ordering, and full reliability.

- Live data applications: concerned with distributing an ongoing flow of small pieces of data. Some examples are stock price distribution and news feeds. Such applications typically require one sender per channel, no start time, varied real-time requirements, varying levels of consistency, varying levels of ordering and reliability.

- Resilient streams applications: concerned with distributing a stream of real-time multimedia data that can handle some losses. Some examples are real-time video and/or audio transmissions. Such applications typically require one sender per channel, various start times, moderate real-time requirements, no consistency requirements, no ordering requirements, and loose reliability requirements.

- Collaborative applications: concerned with sharing data within a group. An example can be a shared whiteboard. Such applications typically require many senders per channel, late join with full data recovery, moderate real-time requirements, various consistency requirements, various ordering requirements, and full reliability.

2.1 Multicast Routing

IP multicast packets are sent to a group of hosts represented by a multicast address independent from physical host addresses. IP multicast sources do not usually know the recipients of the multicast packets. The network layer is responsible for appropriately forwarding data packets to the hosts belonging to the group. Most common multicast routing protocols are: Distance Vector Multicast Routing Protocol(DVMRP)[Pus98], Sparse-mode PIM (PIM-SM) [Est03], Dense-mode PIM (PIM-DM)[ANS03], Multicast Open Shortest Path First (MOSPF) [Moy94], Core-based Tree (CBT)[Bal97] and Distributed Core Multicast (DCM)[BB99].

Two different approaches to the multicast delivery issue exist:

- *source-based* trees (shortest path): rooted at the source and branches form a spanning tree to the receivers;

- *shared* (core-based) trees: rooted at a single common node(Rendezvous Point) placed at a chosen point in the network.

Source-based protocols are designed to find the shortest path through the routing infrastructure between each sender and each of its receivers.

Shared tree protocols attempt to determine a common path which is shared by all senders on a multicast group. Each approach requires some amount of time in order to build the initial routes (tree) between the sender and receivers, and optimize for best path depending on the approach.

DVMRP has shown scalability problems in environments containing more than 8000 subnets, but this limitation can be overcome by a more scalable multicast routing protocol as BGMP [SRT+98].

The primary advantage of source-based trees is that of setting up the optimal path between the source and the receivers. On the other hand maintaining path information for each source is a costly price to pay and does not scale well because of memory requirements. Shared trees main advantage is that of requiring less state "maintenance" information in each router, but the placement of the Rendezvous Point (RP) is critical in order to keep efficiency high.

Since multicast group membership can be very volatile, distribution trees have to be dynamically updated. In fact in case all active receivers on a router branch exit from a particular group, that router must stop forwarding packets through that link, i.e. a part of the tree is cut apart from the rest (pruning). On the other hand a new request for group membership may come from a link where packets were not forwarded before. In this case one or more new branches are added to the multicast delivery tree.

In fact, we can make another distinction between intra-domain and inter-domain multicast protocols. While PIM-SM is the most widely accepted multicast intra-domain routing protocol, inter-domain multicast routing presents some issues.

DVMRP is the oldest dense-mode protocol and is designed to work within an autonomous system so it does not scale to large Internet-wide applications. Although DVMRP is an old broadcast and prune protocol it is a very efficient distributed protocol for multicast packet delivery. DVMRP can tunnel multicast packets via unicast packets, which makes it possible to route multicast packets over unicast networks. The multicast prototype network MBONE [Kre95] has made a lot of use of this feature.

MOSPF is a dense mode routing protocol that uses link-state information to calculate its routes through network.

DCM is a core-based multicast routing protocol for many groups with few receivers. DCM utilizes multiple cores per group to avoid the problem of selecting the optimal position of the single core. DCM is intended for use within a large single Internet domain network with a very large number of multicast groups with a small number of receivers, for example, when multicast addresses are allocated to mobile hosts. For such cases, existing dense or sparse mode multicast routing algorithms do not scale well with the number of multicast groups. A brief taxonomy of the most important multicast routing protocols is shown in table 2.1, while deeper information is to be found in Almeroth's works on multicast distribution trees [CA03]

2.1.1 Aggregated Multicast Routing

IP multicast makes use of a tree delivery structure on which data packets are duplicated only at fork nodes and are forwarded only once over each link, allowing

Routing Protocol	Shortest Path	Shared Tree	Inter-domain	Intra-domain
DVMRP	X			X
PIM-SM	X	X	X	X
PIM-DM	X			X
MBGP	X		X	
MOSPF	X			X
CBT		X		X
DCM		X		X
QoSMIC	X	X	X	
BGMP	X	X	X	X

Table 2.1: Multicast routing protocol taxonomy

IP multicast to scale and support very large multicast groups. However, a tree delivery structure requires all tree nodes to maintain per-group forwarding information, which increases linearly with the number of groups. Growing number of forwarding state entries means more memory requirement and slower forwarding process since every packet forwarding action involves an address look-up. Multicast scales well within a single multicast group but suffers from scalability problems when the number of simultaneous active multicast groups is very large (the scalability of router tables issue).

To improve multicast state scalability, Cui and Kim proposed *aggregated multicast* [CKM+03], a scheme to reduce multicast state size where multiple multicast groups are forced to share one distribution tree (aggregated tree). This way, the number of trees in the network may be significantly reduced. Consequently, forwarding state is also reduced: core routers only need to keep state per aggregated tree. The trade-off is that this approach may waste extra bandwidth to deliver multicast data to non-group-member nodes. Cui's simulations show that aggregated multicast can achieve state and tree management overhead reduction at a reasonable bandwidth consumption expense.

2.1.2 QoS Routing

Zappala proposed [Zap04] that applications specify a QoS request to the network as some combination of delay, bandwidth, and loss characteristics. The QoS routing protocol then computes a path that has available resources and installs this path for the application. Computing a QoS-capable path in a scalable manner can be very difficult since network conditions may change rapidly. In general, this model works well when the shortest paths are provisioned so that they handle most of the load in the network.

2.1.3 Last Hop Problem

Current IP multicast protocols have been designed assuming universal multicast deployment. This prevents end-systems that are more than one hop away from a multicast-enabled router to participate in multicast. Moreover hubs and switches often exclude some host from the rest of multicast-enabled network because they support outdated versions of multicast protocols. Furthermore, transport layer port numbers are not considered part of the identification of a multicast group. Firewalls and NATs exploit these properties to open a channel for the data stream automatically when seeing the first packets of a unicast flow. Since multicast is signaled as a separate protocol and without transport level port numbers, the Firewall/NAT should be able to understand the messages of the multicast management protocol in order to achieve the same level of automation and transparency.

A solution is given by Hjalmtysson who describes Self-Configuring Lightweight Internet Multicast (SLIM)[HBH03], This single source multicast protocol does not require any multicast specific infrastructure beyond its Topology Management Protocol (TMP), that is run in the control plane of multicast capable routers and employs dynamic tunneling where needed. In SLIM a multicast channel is uniquely identified by the pair S,C where S is the IP address of the sender and C is the source specific channel identifier. Receivers interested in the multicast group must know both S and C, and send control messages addressed to S using normal unicast thus avoiding the need for multicast routing. Other solutions are given in section 3.4

2.2 Reliable Multicast Issues

Plain IP Multicast is unreliable: if any data is lost during transmission, there is no mechanism for repairing this loss. For real-time multimedia, this just causes a momentary glitch. For some other data (like a zip archive file), it renders the data useless. Reliable multicast allows to repair lost or damaged multicast data. When a receiver misses one or more data packets, it can ask for the retransmission of these packets. The problem of reliability in IP multicast has been given many answers but still a final widely-acceptable universal solution has not been found. This is mainly due to the different requirements of the various application making use of multicast.

The challenges of supporting reliable multicast include:

- Performing repairs for lost packets to receivers;
- Avoiding ACK/NACK implosion;
- Controlling Congestion;
- Guaranteeing fairness in the use of network bandwidth;
- Ensuring scalability to large receiver groups;

- Guaranteeing robustness to high loss rates;
- Guaranteeing robustness to high transmission latency(e.g satellite links);
- Ensuring Flexibility, managing heterogeneous sets of receivers (as regards loss rate, bandwidth, latency, etc.);
- Maximizing throughput;
- Handling different types of applications requirements.

Chapter 3

State of the Art

3.1 A Taxonomy

In this chapter the main approaches known to date for reliable multicasting are described. Each multicast protocol class can be viewed as using two windows: a congestion window (*cw*) that advances based on feedback from receivers regarding the pacing of transmissions and detection of errors, and a memory allocation window (*mw*) that advances based on feedback from receivers as to whether the sender can erase data from memory [LGLA98]. A short classification of the most interesting reliable multicast protocols is given in table 3.1 (where the meaning of column names will be made clear in the following). Due to space limitations we will explicitly describe a only a subset of those protocols.

3.1.1 Sender-Receiver Initiated Protocols

A first categorization of reliable multicast approaches is between sender-initiated and receiver-initiated protocols. These are the two main classes for end-to-end reliable multicasting proposed to date. Even though some protocols do not exactly fall in one category, we can describe the two approaches as follows.

In the **sender-initiated** approach, the sender maintains the state of all the receivers to whom it has to send information and from whom it has to receive acknowledgments (ACKs). Each sender's transmission is multicast to all receivers; for each packet that each receiver obtains correctly, it sends a unicast ACK to the sender. The sender-initiated reliable multicast protocol requires the source to receive ACKs from all the receivers before releasing memory for the data associated with the ACKs. This scheme suffers from the ACK implosion problem.

In contrast, in the receiver-initiated approach, each receiver informs the sender of the information that is in error or missing; the sender multicasts all packets, including retransmissions, and a receiver sends a negative acknowledgement (NACK) when it detects an error or a lost packet.

The combination of ACKs and NACKs has been used in the past for reliable unicast and multicast protocols (e.g. NETBLT [CL87])

The sender-initiated Negative Acknowledgments with Periodic Polling (**NAPP**) protocol [LLGLA96] is a broadcast protocol that groups large partitions of the data that are periodically ACKed, while lost packets within the partition are NACKed. NAPP advances the *cw* by NACKs and periodically advances the *mw* by ACKs. Since NACKs can cause a NACK implosion at the source, NAPP uses a NACK avoidance scheme.

The main limitation of sender-initiated protocols is the need for the source to process all of the ACKs and to know the receiver set. The two known methods that address this limitation are:

1. using NACKs instead of ACKs;
2. delegating retransmission responsibility to members of the receiver set organized into a tree or ring.

Receiver-initiated protocols place the responsibility for ensuring reliable packet delivery at each receiver. The receivers send NACKs back to the source when a retransmission is needed. A receiver-initiated protocol has no explicit mechanism for the source to release data from memory (i.e. advance the *mw*), even though its retransmission mechanisms are scalable (i.e. advancing the *cw*). Since receivers communicate NACKs back to the source, receiver-initiated protocols may experience a NACK implosion problem if many receivers detect transmission errors. To remedy this problem, some receiver-initiated protocols adopt NACK avoidance schemes.

The receiver-initiated with NACK avoidance (**RINA**) protocols have been shown to have better performance than the basic receiver-initiated protocol. In RINA, the sender multicasts all packets and state information, giving priority to retransmissions. Whenever a receiver detects a packet loss, it waits for a random time period and then multicasts a NACK to the sender and all other receivers.

When a receiver hears a NACK for a packet that it has not received and for which it has started a timer to send a NACK, the receiver behaves as if it had sent the NACK(NACK avoidance/suppression). The expiration of a timer without the reception of the corresponding packet is the signal used to detect a lost packet. With this scheme, hopefully only one NACK is sent back to the source for a lost transmission for an entire receiver set. Nodes farther away from the source might not even need to request a retransmission.

In the receiver-initiated approach NACKs from receivers are used to advance the *cw*, which is controlled by the receivers, and the sequence number in each multicast session message is used to ”poll” the receiver set, i.e. to ensure that each receiver is aware of missing packets. RINA protocols have some limitations:

- a RINA protocol requires the application to provide data for retransmissions. However, this approach does not always work for multimedia applications.

- NACKs and retransmissions must be multicast to the entire group in order to allow NACK suppression. The basic NACK avoidance algorithm requires that timers be set based on the size of the set of receivers. As the number of nodes increases, each node must do increasing amount of work.

Pingali [PTK94] and Levine [LGLA98] showed that receiver-initiated protocols are more scalable than sender-initiated protocols, because the maximum throughput of sender-initiated protocols is dependent on the number of receivers, while the maximum throughput of receiver-initiated protocols becomes independent of the number of receivers as the probability of packet loss becomes negligible.

However, the ideal receiver-initiated protocols cannot prevent deadlocks when they operate with finite memory, i.e. when the applications using the protocol services cannot retransmit any data themselves. Receiver-initiated protocols can have inherent scaling limitations due to the need to set up timers for NACK avoidance, the need to multicast NACKs to all hosts in a session, and the need to store all messages sent in a session.

3.1.2 Tree-based Protocols

Tree-based protocols split the set of receivers into groups, distributing retransmission responsibility over an acknowledgement tree (ACK tree) structure built from the set of groups, with the source as the root of the tree.

The ACK tree includes the receivers and the source organized into local groups, with each such group having a group leader in charge of retransmissions within the local group. Each group leader other than the source communicates with another local group closer to the source to request retransmissions of packets that are not received correctly. Group leaders may be children of another local group, or may just be in contact with another local group [LP96]. Each local group may have more than one group leader to handle multiple sources. Group leaders could also be chosen dynamically. The leaves represent hosts that are only children. It is interesting to note that an ACK tree consisting of the source as the only leader and leaf nodes corresponds to the sender-initiated scheme.

Acknowledgments from children in a group, including the source's own group, are sent only to the group leader. The children of a group send their acknowledgements to the group leader as soon as they receive correct packets, advancing the *cw*. We refer to such acknowledgements as local ACKs or local NACKs, i.e., retransmissions are triggered by local ACKs and local NACKs unicast to group leaders by their children.

Tree-based protocols can also delegate to sub-trees leaders the decision of when to delete packets from memory (i.e. advance the *mw*), which is conditional upon receipt of aggregate ACKs from the children of the group. Aggregate ACKs start from the leaves of the ACK tree, and propagate towards the source, one local group at

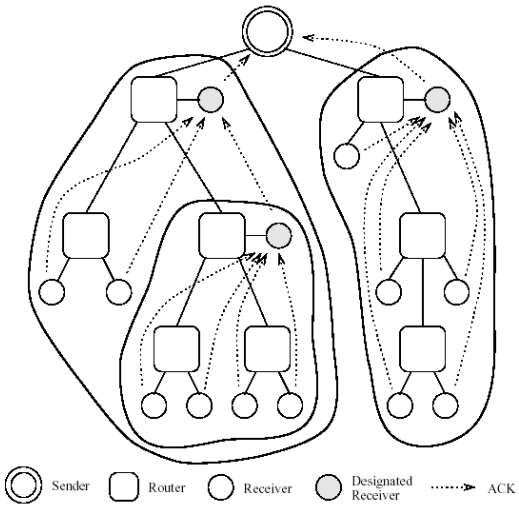

Figure 3.1: **Example tree-based protocol**

a time. A group leader cannot send an aggregate ACK until all its children have sent an aggregate ACK. Using aggregate ACKs is necessary to ensure that the protocol operates correctly even if group leaders fail, or if the ACK tree is partitioned for long periods

The use of local ACKs for requesting retransmissions is important for throughput. If the source scheduled retransmissions based on aggregate ACKs, it would have to be paced based on the slowest path in the ACK tree. Instead, retransmissions are scheduled independently in each local group.

Tree-based protocols eliminate the ACK-implosion problem, free the source from having to know the receiver set, and operate only on messages exchanged in local groups (between a group leader and its children in the ACK tree). Furthermore, if aggregate ACKs are used, a tree-based protocol can work correctly with finite memory even in the presence of receiver failures and network partitions [LGLA98].

One of the first tree-based protocol is the Reliable Multicast Transport Protocol (RMTP)(see fig 3.1) [LP96]. While a generic protocol sends a local ACK for every packet sent by the source, RMTP sends local ACKs only periodically, so as to conserve bandwidth and to reduce processing at each group leader, increasing the effective throughput.

A *tree-NAPP* protocol is defined as a tree-based protocol that uses NACK avoidance and periodic polling in the local groups. NACKs alone are not sufficient to guarantee reliability with finite memory, so receivers send a periodic positive local ACK to their parents to advance the *mw*. An example tree-NAPP protocol is TMTP [YGS95].

Protocol	Send-in	Rec-in	ACK	NACK	Suppr	Aggr	Tree-b	Ring-b
SRM		X		X	X			
LRMP		X		X	X			
RMTP	X		X			X	X	
MFTP		X	X	X	X	X		
TRAM		X		X			X	
TMTP	X		X	X	X	X	X	
NORM		X		X	X		X	
NAPP	X		X	X	X			
RMP	X		X	X	X	X		X
ARM		X		X	X	X	X	
DyRAM		X		X	X	X	X	

Table 3.1: Reliable multicast protocol taxonomy

3.1.3 Ring-based Protocols

Ring-based reliable multicast protocols have only one token site responsible for acking packets back to the source. The source times out and retransmits packets if it does not receive an ACK from the token site within a timeout period, otherwise it advances its *cw*. The ACK also timestamps packets, so that all receiver nodes have a global ordering of the packets for delivery to the application layer. In fact the protocol does not allow receivers to deliver packets to the upper layer until the token site has multicast its ACK, and this may increase latency.

Receivers send NACKs to the token site for selective repeat of lost packets that were originally multicast from the source. The ACK sent back to the source also serves as a token passing mechanism. If no transmissions from the source are available to piggyback the token, a separate unicast message is sent.

The token is not passed to the next member of the ring of receivers until the new site has correctly received all packets that the former site has received. Once the token is passed, a site may remove packets from memory. The final deletion of packets from the collective memory of the receiver set is decided by the token site, and is conditional on passing the token. The source deletes packets (i.e. advances its *mw*) only when an ACK/token is received [LGLA98].

3.2 Multiple Rate

The paths from the server to the clients are usually heterogeneous in bandwidth availability. Multiple-Channel Multicast (MCM)[DAZ99] is used to handle this heterogeneity.

According to this approach data is multicast over multiple channels, each ad-

dressed as a separate multicast group . Each receiver subscribes to a set of channels (i.e. joins the corresponding multicast groups) commensurate with its own rate capabilities (layered multicast). Of particular interest in the design of MCM schemes is the scheduling of data transmission across the multiple channels to accommodate asynchronous requests from clients. MCM schemes select the number of channels, the rate per channel and decide a schedule for transmission of the data over the channels. The basic form of MCM involves sending the entire file over separate multicast channels at different rates. A receiver joins the channel that has the highest rate that does not exceed the receiver's rate capability. This Destination-Set Splitting approach has the disadvantage that a lot of redundant data is sent over each channel. Furthermore, since each receiver listens to only one group, the number of receivers per group may be limited.

Bhattacharyya et al. [BKTN98] propose scheduling the data across multiple multicast groups with receivers joining multiple groups to get the entire file. If the scheduling across the groups is done well, this is more efficient than Destination-Set Splitting. In this Static Striping approach, the groups and rates are scheduled a priori after all receivers register their rates with the sender. Using registered rates, optimal data rates and schedules for a set of channels can be derived. This approach suffers from the problem of requiring the collection of the receiver rates at the sender, which limits scalability with respect to the maximum number of simultaneous receivers. In addition, receiver startup must be synchronized since transmission is tailored for all receivers to start together.

Consider a file that is made up of four fixed-size pieces: A, B, C and D. The sender has enough outgoing bandwidth to transmit four packets per time unit, and there are three receivers, R1, R2, and R3 capable of receiving one, two and seven packets per time unit, respectively. The sender will use multiple-channel multicasting to accommodate the disparity in receiver rates, ensure that each receiver gets all the packets in a reasonable amount of time, and support receivers that join the session at arbitrary times.

One option for multiple channel multicasting is for the sender to use three channels (corresponding to separately addressed multicast groups G0, G1 and G2) operating at rates of one, one, and two packets per time unit, respectively. Fig. 3.2 [BKTN98] shows one possible method for scheduling the packets over the three channels. The advantage of this particular schedule is that receivers can join different multicast groups according to their reception rate capability. Each receiver gets all four packets of the file in a reasonable amount of time, given the receiver rate constraints.

As we have seen above, the design of a multiple-channel multicast schedule must specify four components:

1. number of channels;

2. rate per channel;

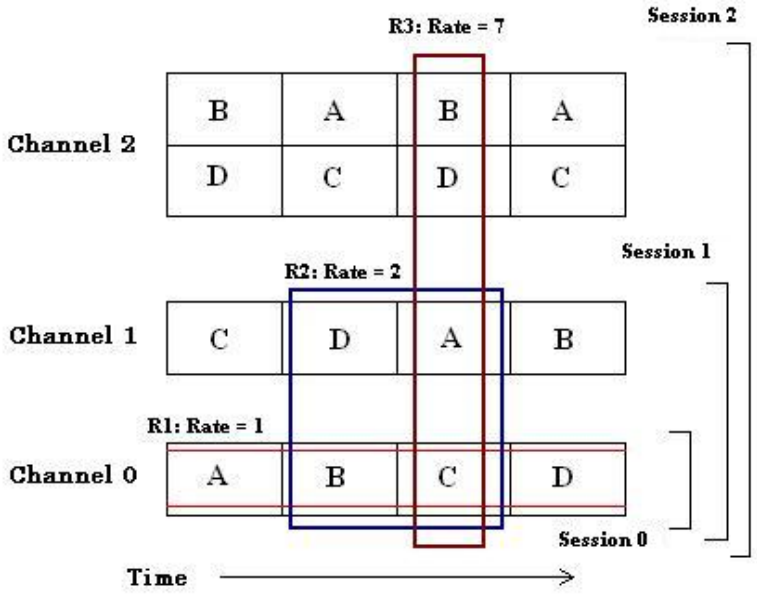

Figure 3.2: **Example efficient multirate transmission schedule**

3. schedule for packet transmissions on each channel;

4. policy for receivers to join channels.

The multiple-channel multicasting approach can handle a large number of heterogeneous receivers because each receiver joins channels based on its own rate, without requiring the sender to worry about flow control information.

3.3 Multiple Channel

Kasera's proposal [KHTK00] involves the use of multiple multicast groups for reducing receiver processing costs in a multicast session. In his proposal a single multicast group is used for the original transmission of packets.

Only retransmissions of packets are done to separate multicast groups, which receivers dynamically join or leave. Kasera shows that protocols using an infinite number of multicast groups incur much less processing overhead at receivers compared to protocols that use only a single multicast group. This is due to the fact that receivers do not receive retransmissions of packets they have already received correctly. The minimum number of multicast groups required to keep the cost of processing unwanted packets to a sufficiently low value can be calculated.

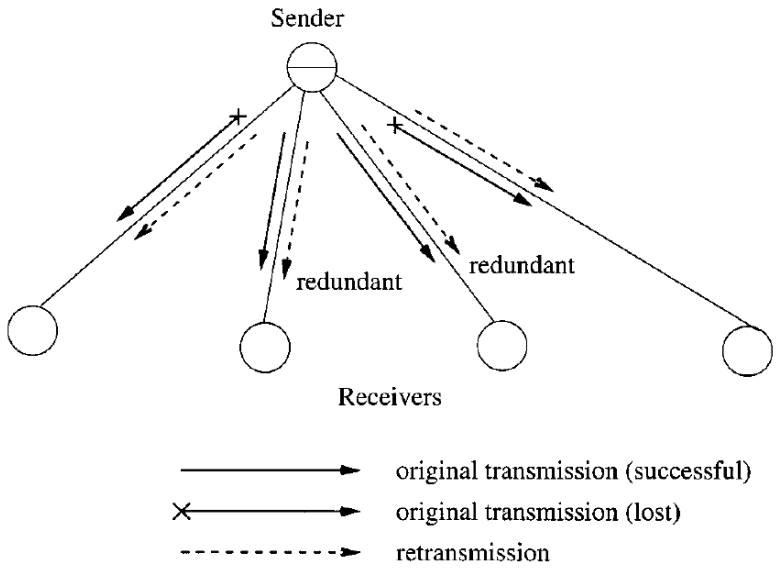

Figure 3.3: **Unnecessary transmissions**

For an application consisting of a single sender transmitting reliably to many receivers only a small number of multicast groups are required for a wide range of system parameters. Furthermore a local filtering scheme for minimizing join/leave signaling when multiple multicast groups are used is shown.

To illustrate the problem in using a single multicast group, consider a reliable multicast scenario with a single multicast group. Since all packet transmissions and retransmissions are done using the single multicast group, each receiver receives all the retransmissions of a packet even after correctly receiving the packet. This can impose unnecessary receiver processing overhead especially as the number of receivers increase (see fig. 3.3 [KHTK00]).

If a number of retransmission groups are used, the receiver will join only the channel(s) of interest (it knows where the retransmission will happen based on the offset of the missing packet), and leaves them after having received the retransmission.

A concern with using multiple multicast groups is the processing of ”joins” and ”leaves”, as receivers dynamically add and delete themselves from these groups. In order to avoid shifting the load from the receivers to the underlying network and at the same time maintaining the reduced processing benefits, Kasera proposes a receiver filtering mechanism that can result in the same amount of network traffic as in the case of a single multicast group, but does not require joins and leaves to be processed inside the network once the multicast session begins.

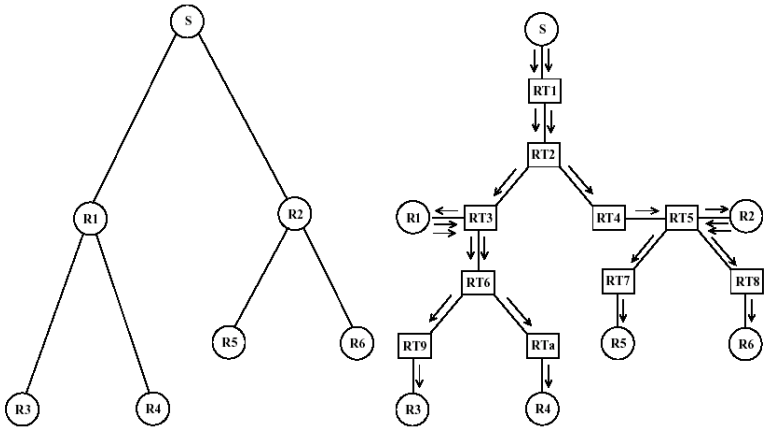

Figure 3.4: **Application-Level Multicast.**

Local filtering can be done by the receiving host or by the network infrastructure. In the first case receiver processing cost is reduced but bandwidth usage is the same with respect to classic recovery schemes. Router cooperation can help reduce bandwidth usage.

3.4 Application-Level Multicast

A large number of multicast protocols assume universal connectivity between end hosts, as we have seen in 2.1.3. However, in reality, this assumption is not valid because of the widespread use of NATs and firewalls. It is important to consider connectivity restrictions because NATs and firewalls are heavily deployed, affecting the proper functionality and the performance of multicast protocols [LCG04].

Motivated by the desire to spark the growth of multicast use, and frustrated by the lack of widely available network layer multicast, some researchers have concluded that multicast should first be realized as application layer overlays (ALM or AGCS) [CDKR02].

Figure 3.4 [ES04] shows an example physical topology and its overlay multicast layout.

Application-level multicast sends data using unicast, so flow and congestion control as well as reliable delivery of traditional unicast services can be taken advantage of. On the other hand bandwidth saving is less than using actual multicast technology.

An interesting research work has been done by Parnes [PSS98] which examines a lightweight application level multicast tunneling using mtunnel. Su-Wei aims to lower application delay of application-layer multicast [SWW03] whereas Mathy reviews the security issues of this kind of multicast [MBRES04].

3.5 Forward Error Correction

Forward Error Correction is a technique adding redundant information in the transmitted data flow, allowing a receiver to recover the original information even under partial loss. FEC in multicast connections can have a big effect, since the global loss will increase with the number of receivers. The drawback to FEC is the complexity of encoding and decoding.

The effect of FEC on the multicast tree is strongly dependent on shared links. Nearly all the research on the performance of reliable multicast communication assumes multicast trees with links where the loss over any link affects only a single receiver, referred to as independent loss. The assumption that the paths to different receivers have no shared links is typically not true for trees constructed by multicast routing algorithms as CBT or PIM.

Main results by [CGI+99] are: When FEC is applied to all links of the multicast tree, for small groups the FEC gain increases as the number of receivers increases and the number of shared links decreases. For big groups the FEC gain is larger than for small groups and is independent of the number of receivers.

When FEC is applied either on the links of the LAN part or either on the links of the WAN part of the multicast tree, it can be observed that for small groups the highest FEC gain is achieved when applied on the part where the links have a high loss probability. For big groups this is only true for a low number of receivers that can be found in one LAN. For a large number of receivers in one LAN, FEC should be applied on the LAN links. Canetti also shows that the assumption of independent loss among receivers is a worst case assumption.

3.6 A Survey of Reliable Multicast Protocols

3.6.1 SRM

Scalable Reliable Multicast [FJL+97] provides a good solution for NACK implosion, distributes loss recovery to all members in the group, and is robust with respect to changes in group membership or topology. However, its timer-based implosion control mechanism increases recovery latency, and it has problems with duplicate requests and repairs. Additionally, local recovery is still an open issue for SRM. SRM relies on topology information to set its timer values, and hierarchical approaches require receivers to locate their designated representatives.

3.6.2 LRMP

The Light-weight Reliable Multicast Protocol [Lia98] offers reliable and source ordered data delivery service for group communications. It adopts a random expanding probe scheme for local error recovery so that no prior configuration and no router

support are required. Subgroups are formed implicitly and have no group leaders. Packet loss is reported upon a random timeout incrementally until it is repaired. This simple scheme is used for duplicate NACK and repair suppression.

3.6.3 TRAM

The Tree-based Reliable Multicast protocol [CHKW98] is designed to support bulk one-to-many data transfer. It uses dynamic trees to implement local error recovery and to scale to a large number of receivers. It also includes rate-based flow control and congestion control based on feedback from receivers. TRAM uses a selective acknowledgement mechanism for reliability and a hierarchical tree-based repair mechanism for scalability.

The hierarchical tree overcomes implosion-related problems, enables localized multicast repairs and funnels feedback to the sender. The receivers and the data source of a multicast session in TRAM interact to dynamically form repair groups, linked together to form a tree with the sender at the root.

Every repair group has a receiver that acts as a group head; the rest act as group members. Every repair group head in the system is a member of some other repair group. All members receive data multicast by the sender. The group members report lost and successfully received messages to the group head using a selective acknowledgement mechanism similar to TCP's own. The repair heads cache every message sent by the sender and provide repair service for messages that are reported as lost by the members.

The acknowledgement-reporting mechanism is window-based. The flow control is done via an adaptive rate-based algorithm. The sender senses and adjusts to the rate at which the receivers can accept the data. Receivers that cannot keep up with the minimum data rate can be dropped from the repair tree.

Acknowledgements sent by receivers contain a bitmap indicating received and missing packets, which are later repaired by the repair head;

Limitations:

- Realistic TTL Values: tree formation in TRAM relies on TTLs. Current multicast routing protocols may not provide accurate TTL values for this purpose.
- Heterogeneous Receiver Population: a slow receiver may end up without repair service if it cannot keep up with the minimum speed specified by the sender.

A Java implementation of TRAM can be found in JRMS [RKH98].

3.6.4 NORM

The NACK Oriented Reliable Multicast protocol [AB04] is designed by IETF to provide end-to-end reliable transport of bulk data objects or streams over generic IP multicast routing and forwarding services.

NORM uses a selective NACK mechanism for transport reliability and offers additional protocol mechanisms to conduct reliable multicast sessions with limited ”a priori” coordination among senders and receivers.

NORM's congestion control scheme allows to share available network bandwidth with other transport protocols such as TCP.

NORM is capable of operating with both reciprocal multicast routing among senders and receivers and with asymmetric connectivity (possibly a unicast return path) from the senders to receivers. NORM uses FEC repair and other IETF reliable multicast transport building blocks in its design.

3.6.5 ARM

Active Reliable Multicast [LGT98] is an active protocol in that routers in the multicast tree play an active role in loss recovery.

ARM uses a receiver-based, NACK-based scheme. Receivers detect losses primarily by sequence gaps in the data packets. A receiver sends a NACK towards the sender as soon as it detects a loss.

In the upstream direction, routers suppress duplicate NACKs from multiple receivers to control the implosion problem. By suppressing duplicate NACKs, ARM also lessens the traffic that propagates back through the network. In the downstream direction, routers limit the delivery of repair packets to receivers experiencing loss, thereby reducing network bandwidth consumption. ARM-compatible routers cache multicast data on a ”best-effort” basis.

Multiple NACKs from different receivers are cached and aggregated at active routers along the multicast tree. The sender responds to the first NACK by multicasting a repair to the group. It then ignores subsequent NACKs for this packet for a fixed amount of time

ARM caches multicast data packets at routers in the multicast tree for possible retransmission. Any router along a NACK path can retransmit the requested packet if it has that packet in its cache.

Active routers scope retransmission of repair packets to the portions of the multicast group experiencing loss. They do this by looking up the corresponding subscription bitmap, which was created and left in the router's soft-state cache by previous NACKs.

In comparison with SRM, ARM has a much lower recovery latency and provides a specific solution (router-based scoped retransmission) to local recovery.

ARM is in theory more flexible and robust than the hierarchical approaches, because any router with cached multicast data can perform retransmissions, and end hosts do not have to maintain nor have any knowledge of group topology. On the other hand, the main advantages are obtained when most of the routers are ARM-capable, but this also implies a heavier computational and storage load at routers.

3.6.6 DyRAM

DyRAM is a reliable multicast protocol whose recovery strategy is tree-based with cooperation from DyRAM-capable multicast routers and works on a per-packet basis [MP02]. DyRAM receivers are responsible for loss detection and signaling, but are also capable of retransmitting the repair packets (local recovery). As for ARM, routers play an explicit role: they suppress duplicate NACKs and subcast repair packets only through interfaces leading to hosts that have experienced that packet loss. One main difference from ARM is that DyRAM routers can elect, for each single packet loss, an interface wherefrom local recovery will be attempted. Furthermore DyRAM does not by default cache transmitted data at routers. The per-packet recovery strategy of DyRAM produces interesting results with respect to the load at the source and the bandwidth consumption, but has disadvantages in terms of security and load at DyRAM routers.

3.6.7 MFTP

MFTP [Ma98] is a receiver-initiated multicast transport protocol optimized for bulk data transfers.

MFTP uses a rate-based, NACK-based scheme and the sender is almost constantly sending data and very seldom has to wait for acknowledgements from the receivers. This makes MFTP especially suited for transmissions over links with long delays, for example satellite links.

The protocol behavior can be summarized in the following way: the data sender, MFTP server in what follows, manages the multicast groups, initiates the file transfer and controls the transfer operation. The data receiver, MFTP client in the sequel, joins the multicast group, receives the data sent by the MFTP server and provides reception status to the server when requested. The protocol consists of two parts: an *administrative protocol* which deals with groups and session management; *a data transfer protocol* which deals with the simultaneous, reliable transmission of a file from an MFTP server to multiple MFTP clients.

Three different use cases are supported by MFTP: *Closed Group* - the MFTP server knows in advance the set of clients allowed to participate to a session; *Open Limited Group* - any MFTP client interested in participating in the data transfer has to register with the MFTP server, which may in turn limit the number of participants; *Open Unlimited Group* - any MFTP client is allowed to participate in the data transfer. In the latter case MFTP clients are not required to register with the MFTP server.

File Transfer is accomplished through three different phases:

1. *File Announcement*: the MFTP clients are informed that a file is about to be transmitted and on the transmission characteristics;

2. *File Delivery*: the file is transmitted by the MFTP server to the MFTP clients in the multicast group;

3. *File Completion*: both the clients and the server are informed that file transfer has been completed.

We will describe now the file delivery phase in more detail. The file is initially segmented by the MFTP server into blocks containing a fixed number of equally sized Data Transfer Units (DTUs).

Each datagram sent by the server contains a single DTU and indicates its position in the file (i.e.the DTU and block number are part of the MFTP packet header). In this way each transmitted DTU can be univoquely identified.

The file is transmitted in steps. In the first step the whole file is transmitted. In subsequent steps the server only retransmits the DTUs which were reported as missing by some MFTP client. Only missing DTUs are retransmitted. Retransmissions are repeated until the file has been successfully delivered to all MFTP clients.

MFTP clients are queried by the MFTP server for their receive status at the end of each block transmission. A client keeps track of the received and missing DTUs by updating a data structure composed of an array of m bitmaps, where m is the number of blocks the file is segmented into. A client that has not received some DTU sends a response message containing a bitmap (where each bit represents the reception status of a DTU within a block). Such response (NACK) message contains the block number in its header, whereas the payload is the above-mentioned bitmap. In this way the reception status of an entire block of DTUs (1000s of datagrams) can be transmitted using just a single datagram (NACK compact format).

Main features that enhance MFTP scalability are: NACK suppression, aggregation and compact format.

Chapter 4

Main Contribution

4.1 JMFTP

This section reports on our research work and on the main results of investigating the potentiality of Java in the design and implementation of reliable multicast protocols. Mathematical analysis by itself does not suffice for evaluating complex multicast behavior. Network simulation is a promising approach but care must be taken in the design of network models that reflect real network behavior. Actual experimentation of protocol implementations can be useful for gathering valuable data for evaluation purposes.

In this section we focus on the design, implementation and performance evaluation of JMFTP, a Java implementation of MFTP. JMFTP is the acronym for Java Multicast File Transfer Protocol. It pursues the following goals:

- simplicity and effectiveness;
- no performance change over high delay networks;
- flexibility, making it possible to invoke optional features, such as response aggregation and/or suppression, to increase scalability.
- feasibility, providing a framework for building multicast applications;
- scalability to thousand of receivers over one hop (e.g.satellite) networks with no intermediate relaying entities;
- portability for multi-platform support;

4.1.1 Architecture and Software Implementation

JMFTP consists of the following three functional entities: JMFTP server, JMFTP client, JMFTP relay. The main purpose of the latter is to limit the amount of control

packets the server receives from the clients, in order to prevent possible flooding phenomena, thus increasing scalability.

The JMFTP architecture has distinct modules:

Sending modules:

- Product Sender - fetches data from a *data to send pool* structure, encapsulates data into JMFTP data packets and multicasts the packets. It is also in charge of sending JMFTP response request packets;

- Response Sender - fetches responses from a *response to send pool* structure, encapsulates responses into JMFTP response packets and multicasts the packets;

- Response Aggregator Sender - fetches responses from a *response to send pool* structure, aggregates responses into JMFTP response packets, and multicasts the packets.

Receiving modules:

- Product Receiver - waits for JMFTP data or response request packets and processes the packets according to their type: the content of a data packet is saved; a response request packet initiates a new response sending phase;

- Response Listener - waits for JMFTP response packets and updates the *response to send pool*;

- Response Receiver - waits for JMFTP response packets and adds to the *data to send* pool the DTUs that, according to the processed response, must be retransmitted.

Figure 4.1 shows the JMFTP software architecture.

Each module is implemented as a Java Thread. All threads meet the Java Thread Design Guidelines, to avoid deadlocks [Sof].

JMFTP implements the following enhanced features:

- on demand disk space reservation at the client side;

- traffic shaping of the flows generated at the server side, in order to avoid congesting the network;

- source code instrumentation, to allow performance measurements and collection.

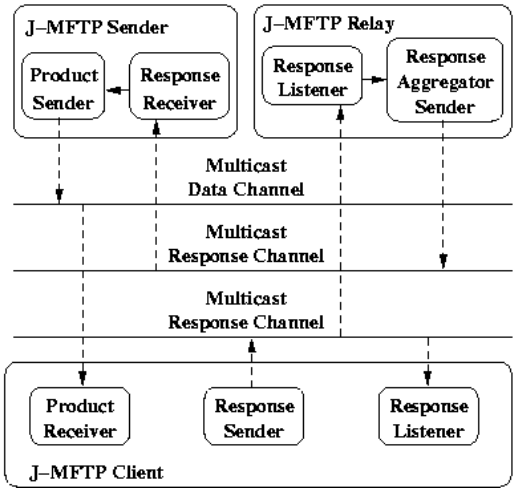

Figure 4.1: JMFTP software architecture.

4.1.2 Java Issues

One major obstacle to the global diffusion of (killer) multicast applications is that software must be separately coded, compiled and deployed to run on different platforms. Java can be the solution to this problem: programs written in Java can be run wherever the Java platform is present. Moreover, Java is mobile, i.e. Java applications can be on demand downloaded and executed as applets. Dealing with high performance requirements in Java is a challenging issue, due to the fact that Java is a partially interpreted, pure object oriented language. Main challenges in implementing a protocol in Java are:

- Java is a partially interpreted language, thus introducing potential inefficiency. This problem could have a dramatic impact on both the reliability of packet delivery (packets can be lost due to software and hardware bottlenecks) and the efficiency in the use of network resources;

- Java is a well-defined object oriented language, so that the designer is continuously faced with a delicate and inevitable tradeoff between good structure and good performance;

- a Java implementation, for its nature, is poorly integrated into the host operating system.

4.1.3 Evaluation

In this section we provide an overview of the testing scenarios in which JMFTP is evaluated and of the results that are achieved.

We are mainly interested in testing the following major aspects of JMFTP behavior: i)the correctness with respect to MFTP functional specifications; ii)the robustness with respect to different network configurations and load conditions; iii)the efficiency achievable over high performance networks.

We also test the tolerance of the implementation with respect to delays in packet delivery (e.g. this may be the case in a satellite link [Bos99]) and loss.

Test environment. JMFTP has been tested on a geographical multicast network interconnecting four LANs that include a dozen of heterogeneous servers and workstations. The machines involved in the experimentation varied from small workstations to server machines equipped by 64 bits, 400 MHz processors and 512 MBs RAM, running Solaris, Digital Unix, Linux and Windows NT.

Performance Metrics. At the moment, the following performance indexes are considered as significant with respect to our goals:

- *Host Transfer Time*(HTT): the time elapsed, for a client, between the reception of the first and of the last DTUs;
- *Total Transfer Time*(TTT): the time elapsed, for the server, between the transmission of the first DTU and the reception of the message attesting that the last client involved in the transmission successfully received the whole file;
- *Average percentage of useless DTUs*: a DTU is useless if it is received by a client which has already received it. Useless DTUs cause bandwidth waste and degrade system performance.
- *Number of lost DTUs*: the number of DTUs that were not received by the client, included retransmitted DTUs.
- *Number of NACKs*: the number of NACKs sent by a client. When compared to the number of lost DTUs, this metric tests the effectiveness of aggregated NACKs.

Test plan and considered scenarios. Starting from one server and one client connected on the same LAN, we construct more complicated scenarios with dozens of clients connected to different LANs. The behavior of JMFTP is evaluated in the following target network scenarios:

- low-bandwidth congested networks with high packet loss rate;
- high-speed networks;
- satellite networks.

The robustness of JMFTP is evaluated over congested low bandwidth networks by varying the following parameters:

- *Loss rate*: packet loss rate is artificially introduced and varied by discarding packets at each client, ranges from 5% to 50%, in steps of 5%;
- *Data rate*: maximum transmission rate in bits per second. The following values are considered: 57,600 bps, representing a typical modem connection via an analog link; 115,200 bps, modeling an ISDN telephone line; 512, 1,024, 2,048 and 4,096 Kbps, which are typical values for enterprise networks;

The performance of JMFTP over high speed networks is metered in the following conditions:

- *Loss rate*: packet loss rate is not artificially induced;
- *Data rate*: maximum transmission rate, in bits per second, ranges from 100 Kbps to 10Mb/s over 100Mb/s physical networks;

The robustness of JMFTP over satellite networks is metered in the following conditions:

- *Loss rate*: packet loss rate is not artificially induced;
- *Data rate*: the maximum transmission rate is about 2 Mb/s;
- *Delay*: a fixed transmission delay of 500 ms is artificially induced in the network.

The first problem we deal with is generating a possibly isochronous flow of packets corresponding to a given nominal throughput. This can be crucial since excessive burstiness may lead to undesired congestion of network resources and to overwhelming of clients.

Unfortunately, isochronism is not easily achievable for many reasons, among which the most significant are: i) the Java Virtual Machine is inaccurate in the temporal management of concurrent threads, in particular in suspending their execution; ii) the time spent in accessing secondary storage is not constant and depends on factors that cannot be kept under control; iii) the interaction between the emission thread and other threads, like, for example, the one processing incoming NACKs; iv) the interaction between the emission thread and the operating system. In other words, synchronicity cannot be obtained by simply suspending the thread that emits packets for a constant time after each emission.

Thus, we are induced to adopt the emission strategy described by the Java-like code of figure 4.2: if K Mb/sec is the required throughput, the value of the T parameter is computed as $(dim * 8)/(K * 1000000)$ seconds, where dim is the number of

```
starting _time = system_time;
packet_number = 0;

while (true)
    {
    emission_time = starting_time + (packet_number*T);
    sleeping_time = emission_time - system_time;

    if (sleeping_time > 0)
        sleep(sleeping_time);

    send(packet);
    packet_number = packet_number+1;
    };
```

Figure 4.2: JMFTP:The emission strategy

bytes in a packet. The strategy tries to match the nominal throughput as closely as possible by adapting the time elapsing between two consecutive packet emissions.

The effectiveness of the strategy in realizing different throughputs is tested on three different Hw/Sw platforms: a workstation Sun Ultra 5 (333 MHz)/Solaris 2.8; a personal computer Hewlett Packard (Xeon 500 MHz)/Linux 2.2; a personal computer Hewlett Packard (Xeon 500 MHz)/Microsoft Windows NT 4.0.

With respect to the above issue we consider a simple testing scenario in which a J-MFTP server is connected to a J-MFTP client through a 100Base T Ethernet segment. A third personal computer connected to the LAN segment runs a network monitoring tool.

In this and subsequent tests a 10MB file is transmitted through MFTP packets of size 1.500 bytes.

We measure the average throughput obtained on the different platforms considered for different values of T, and compare the obtained values with the corresponding nominal throughputs. The results of our tests are resumed in figure 4.3: measurements show that both the Sun Ultra 5/Solaris and HP/Linux behave very well up to about 10 Mb/sec. For this emission rate, the actual throughput is 0.4 Mb/sec away from the nominal one, i.e. 9.6 Mb/sec. The HP/Windows also behaves quite well, but it is surely less accurate in achieving the required nominal rate. Furthermore, even skipping the invocation of the *sleep()* method, i.e. letting the system transmit at the highest rate, it is not possible to achieve an average throughput higher than 11.8 Mb/sec.

A second measurement campaign is performed in the same ideal environment described above (one-to-one transmission on a 100 Mb/sec LAN). This has the goal of fixing some reference values on the behavior of different Hw/Sw configurations for

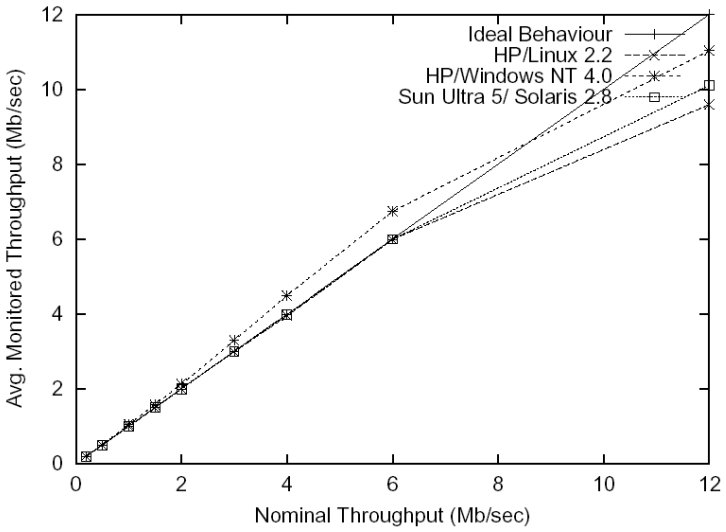

Figure 4.3: **JMFTP:nominal vs monitored emission throughput**

the J-MFTP client, such as the maximum throughput corresponding to the absence of DTU loss and the throughput value that corresponds to the best achievable system performance. These tests also make it possible to evaluate the robustness of the architecture in presence of high throughputs and/or loss rates.

Three Hw/Sw architectures are considered for the J-MFTP client: a workstation Sun Ultra 5 (333 MHz)/Solaris 2.8; a personal computer Compaq (Pentium III 500 MHz)/Linux 2.2; a personal computer Hyundai (Celeron 600 MHz)/Microsoft Windows 2000. Tests are performed for each considered platform and for different values of the throughput. For each test we collect the average values of: i) the host transfer time (HTT); ii) the total transfer time (TTT); iii) the percentage of lost DTUs. To this regard it is worth noticing that, in the considered scenario, with a high probability, DTUs can only be lost by the J-MFTP client because of its inability to sustain the transmission throughput.

Figure 4.4 shows the HTT dependence on the emission throughput. In particular, it is possible to notice that the best performance is not obtained at the highest achievable transmission rate: for both the HP/Linux and the Hyundai/Windows platforms the minimum HTT is reached when the emission throughput is roughly 3.0 Mb/sec. An emission throughput of 4.0 Mb/sec makes the HTT for the Sun Ultra/Solaris architecture as low as possible. For higher throughputs the percentage of lost DTUs considerably increases the number of passes that are necessary to deliver the product, thus also increasing the HTT.

It is interesting to notice that as long as DTU loss is moderate (up to 5%), HTT is

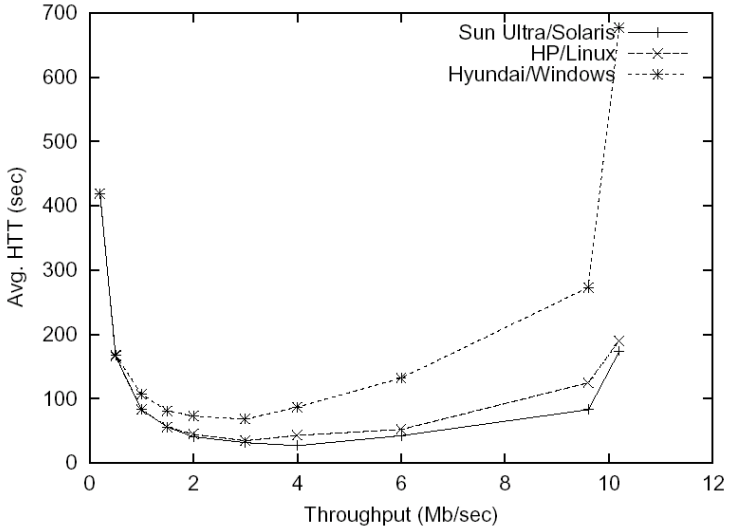

Figure 4.4: **JMFTP:HTT dependence on the emission throughput**

very close to its ideal value, given by the ratio between file size and nominal throughput. Moreover for all considered platforms, HTT remains close to its ideal value for throughputs up to about 4.0 Mb/sec. This shows that the overhead introduced by retransmissions remains low in this range.

Figure 4.6 shows how the HTT is influenced by the percentage of lost DTUs. This picture shows that the best performance is not obtained in the absence of loss. This is due to the trade-off existing between throughput and loss rate (see comments above). For this reason, it is convenient to increase the throughput, as long as loss rate is moderate.

Figure 4.5 shows the relation existing between emission throughput and DTU loss rate. As to this point, it is worth noticing that the system is robust, in the sense that the product is always correctly delivered, even in the presence of high loss rates, up to 95%.

A third series of tests is performed in the same reference scenario in order to evaluate the impact of the loss of DTUs on the HTT. The emission throughput of the J-MFTP server is set, for each considered platform, to the maximum throughput that does not introduce loss of DTUs (see comments above), while several DTU loss rates are artificially introduced at the server itself. In this way we can simulate the loss of DTUs due to queue congestion at routers in a geographical network.

We finally move to a more complex scenario in which the already considered platforms are all connected by the same 100 Mb/sec Fast Ethernet LAN, together with two more J-MFTP clients: a laptop Acer (Celeron 466 MHz)/Linux 2.4 and a

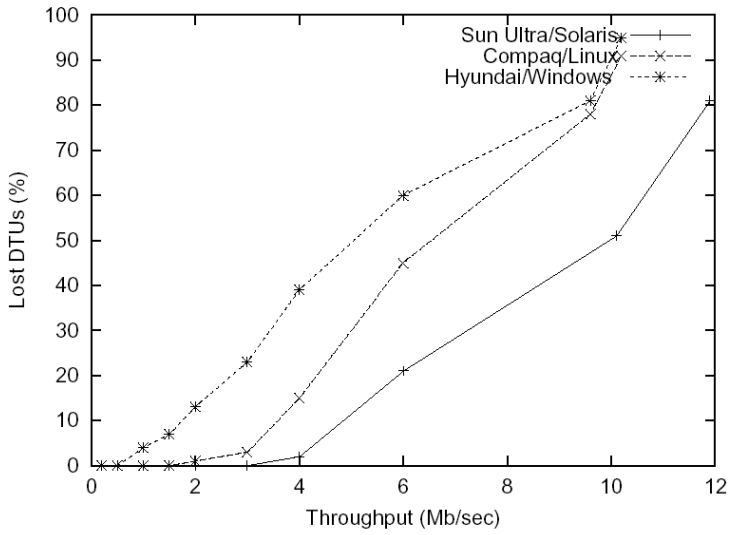

Figure 4.5: **JMFTP:relation between emission throughput and percentages of lost DTUs**

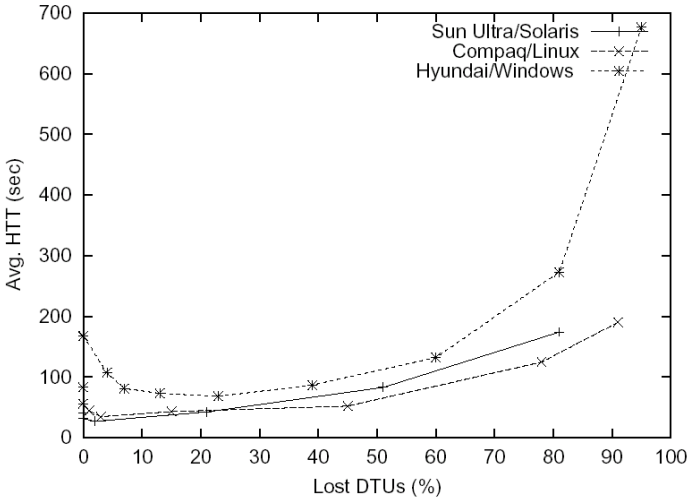

Figure 4.6: **JMFTP:HTT dependence on the DTU loss rate**

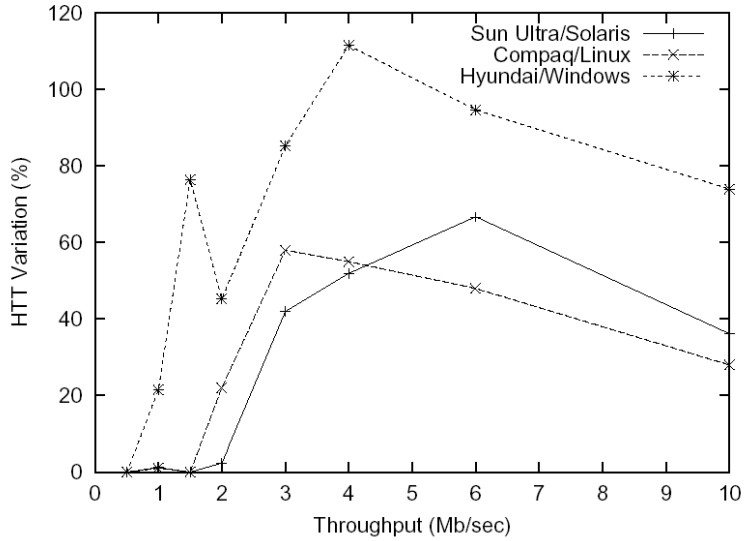

Figure 4.7: **JMFTP:percentage variation of the HTT with respect to the one-to-one case**

laptop Compaq (Pentium II 266 MHz)/Windows 2000.

This scenario is useful to evaluate the impact of the concurrent presence of more clients on the performance of each single client. This influence is mainly due to the fact that each client is interrupted by the reception of the responses (NACK) multicast by all other clients. With regard to this, the user-level performances of each client (i.e. the HTT experienced by a client) are compared to those observed for the same client in the one-to-one case.

Figure 4.7 shows the HTT variation for the three clients considered in this section and for different throughput values. As a general remark, the picture shows that the interaction between clients is considerable. For each considered platform and for every throughput value, the figure shows the percent variation with respect to the value measured in the one-to-one case. Results show that the worsening never exceeds a factor about 1.2, i.e. the HTT of a client when 5 clients receive the product over the same LAN is lower than 2.2 times the value when the client receives the product in a one-to-one configuration. Furthermore:

1. JMFTP has a robust behavior on low-bandwidth network as data rate increases. For example, in the case of a single client of medium/low power (a SUN ULTRA 5 with a 167 MHz processor and 64 MB RAM)and a server connected by the same LAN, the total transfer time decreases as data rate increases up to 4,000 Kb/s. This shows that the client can handle packets efficiently up to a

rate of 4Mb/s.

2. As packet loss rate increases from 0% to 50%, we observe an increase in the total transfer time of roughly an order of magnitude, but the file is always successfully delivered to destination;

3. JMFTP cannot generate isochronous shaped flows with a data rate higher than 4 Mb/s.

Results obtained in the satellite scenario are quite similar to those achieved in presence of negligible delays, thus showing the tolerance of JMFTP with respect to latency.

4.2 Work in Progress

4.2.1 NJMTP

This project extends our reliable multicast software framework with a module written to support and experiment with the following:

- Relay discovery, i.e. the discovery of the tree structure for aggregation and suppression of NACKs;
- Multiple rate;
- Multiple channel;
- Performance tuning for real-time, seeking trade-offs between scalability and timeliness;
- Native thread performance issues (Native Posix Thread Library).

4.2.2 Active Smart Multicast Routing

Starting from previous research on anycast and existing Active Multicast proposals [LGT98] [MP02], we are investigating the active router support for multicast scalability issues. The main goals of this research are:

- Scalability enhancements;
- Performance analysis.

4.3 Conclusion

In this part of the thesis we have surveyed known classes of reliable multicast protocols. Sender-initiated protocols are not scalable because the source must account for every listening receiver. Receiver-initiated protocols are far more scalable if they avoid overloading the source with retransmission requests. On the other hand, these protocols require infinite buffers in order to prevent deadlocks. As a consequence this protocol class can only be used efficiently with application-layer support.

Some solutions to the ACK/NACK-implosion problem are tree-based protocols and ring-based protocols. The first organize the receivers in a tree and send ACKs along the tree; the latter send ACKs to the sender along a ring of receivers.

Tree-based protocols delegate responsibility for retransmission to receivers and employ techniques applicable to either sender-initiated or receiver-initiated protocols within local groups of the tree.

In this context we focused our attention on the performance and scalability of a specific protocol, in order to seek trade-offs among throughput, loss rate and group size. With this aim we investigated the feasibility of a Java implementation of a reliable multicast file transfer protocol (MFTP). We presented a software architecture and proposed experimental settings to test its performance and robustness on different platforms, with respect to a variable number of clients in a real-world environment.

The following conclusions can be drawn with the results presented in this section. The J-MFTP implementation is robust with respect to high packet loss rate conditions. The protocol successfully delivers the file also in presence of a very unreliable channel (even in the presence of packet loss rate up to 95%).

For what regards performance, results show that the bottleneck of the system is at the client side. We also notice that the best performance is obtained when the sender data rate produces a moderate but not negligible loss rate at the client side. This value is highly dependent on the client characteristics.

Along this investigation we have analyzed specific scenarios and gathered experimental data that allows us to contribute to the research community with our results and also by offering a reference high-performance reliable multicast protocol implementation. Results we collected lead us to new research directions that we describe in the second part of this document, such as the impact of security on reliable multicast performance.

Part II

Secure Multicast

Chapter 5

Introduction

Group communication services such as chat, videoconferencing and online gaming are becoming increasingly popular. Pay-TV and video on-demand are at present a part of the networked world. The need for security in these environments is very important especially for the service provider and in some applications for the end user, too.

Security properties are not equally essential in every kind of group application but their need is dependent on the type of trust relationship between communicating parties. For example a pay-TV service provider wants to be sure that program transmissions are viewed by the paying customers only. The various secure multicast scenarios and concerns are quite diverse and even contradictory.

We can distinguish different types of secrecy:

- *Ephemeral secrecy* i.e. preventing non group-members from easy access to transmitted data.
- *Long-term secrecy* i.e. protecting the confidentiality of data for a long period of time.

Usually for multicast data guaranteeing the first type of secrecy is required whereas the second is rarely needed.

In this chapter we will briefly describe secure multicast issues, together with a view of the most interesting proposals to date.

5.1 Multicast Security Issues

Multicast can provide an efficient delivery service to large groups of users, but multicast security has to be taken into account when authorized access to information and control over the set of participants to a multicast session is required. Cryptography can be used to satisfy the requirements of a secured application, such as:

- Confidentiality;
- Integrity;
- Authentication;
- Availability.

Message *confidentiality* can be assured e.g. by using symmetric key encryption for data. On the other hand employing a single key shared by the receivers and the sender cannot prevent the forgery of the packets by a group member.

Data *integrity* can be guaranteed as follows: the sender computes a digest of the message, encrypts the computed digest and includes the encrypted value in the message that is to be sent. Receivers compute the digest of the message and compare the computed digest against the decrypted digest extracted from the received message. Senders use their private key to generate a signature of the data, and receivers use the corresponding public key to verify the signature. Alternatively, shared secret keys may be used, either a separate one for each sender, or one key used by all senders.

Sender *authentication* can be guaranteed by using digest signature with public/private key pair as above.

Service *availability* (i.e. the capacity to detect and resist to Denial of Service (DoS) attacks) is relevant in a multicast setting, since attacks are easier to mount and are much more harmful. Here protection should include multicast routers as well as end-hosts.

The *authentication*, *integrity* and *confidentiality* mechanisms are very similar to those for unicast connections. and can be obtained by MAC (Message Authentication Code) or the digital signature, the message digest with hash function and data encryption, respectively. These solutions are designed to work in unicast environment but generally for secure multicast more scalable solutions are required.

Multicast security concerns are considerably more complex than those regarding point-to-point communication. They can be summarized as:

- Access control;
- Dynamic group membership;
- Trust in routers.

Main parameters that influence secure multicast group management are:

- Group size;
- Member computing power;
- Membership dynamicity.

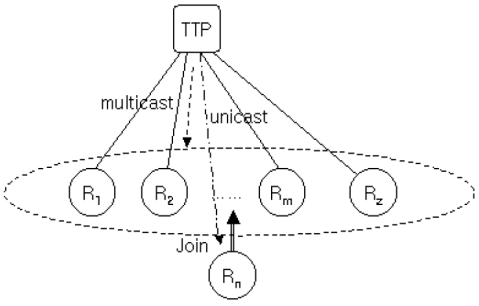

Figure 5.1: **Example secure group join**

Performance is really important for secure multicast applications. The main performance issues regard:

- Latency overhead per sending and receiving packets containing keys;
- Bandwidth overhead in data transmission due to encryption/decryption;
- Group management activity such as group initialization and group membership change.

As regards **multicast routing**, it clearly presents many vulnerabilities, since: i) multicast addresses are publicly known; ii) joining or leaving a multicast group does not require special permissions; iii) an intruder can send data to the group without being a member.

In the following chapter we will briefly describe some of the most interesting proposals for key management, whereas for a single specific solution regarding multicast routing we refer to [SGLA99].

5.1.1 Group Key Management

A common solution for multicast message *confidentiality* is by the adoption of a *session key*(group key), i.e. a symmetric(shared) key that must be simultaneously owned by the sender and all the receivers of the multicast stream. If the service has to be paid for, the group key should be changed on every membership revocation. Every time a member leaves the group a new group key has to be generated and distributed to every group member. The new group key has to be encrypted with member's personal key encryption key (KEK).

If a receiver joins a multicast group (See fig.5.1 [CK00] where TTP=Trusted Third Party), the group key is changed to prevent the new receiver from decrypting the data that was transmitted before. This property is called *backward secrecy*.

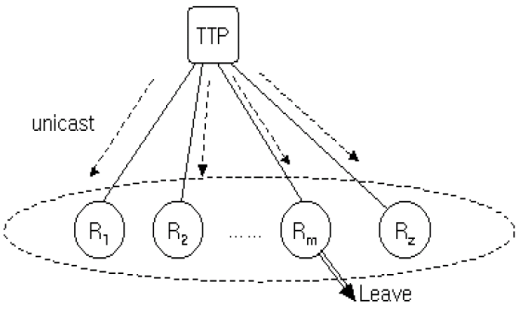

Figure 5.2: **Example secure group leave**

If a receiver leaves a multicast group (See fig.5.2 [CK00]), the group key should be changed also. Otherwise, the leaving receiver can continue to decrypt data using the previous group key. This property is called *forward secrecy*. Guaranteeing backward secrecy is easier, in that the sender can send a new session key to group members via the encrypted multicast channel to the present group.

The above-mentioned operations involve two tasks: key encoding and key distribution. The *key encoding* phase of group rekeying involves generating a set of encrypted keys that have to be transmitted to the members of the group. The *key distribution* (rekey transport) phase is concerned with packing these encrypted keys into packets and delivering the packets to the members of the group. For scalable group rekeying, both the key encoding and key distribution operations need to be scalable.

The most straightforward way to distribute keys is by using a centralized group key controller (GC), which takes care of group key management. In this case the distribution cost is linear to the group size and the controller has to store $n+1$ keys. This simple key management unfortunately does not scale well and has the disadvantage of having a single point of failure.

In fact for large groups with frequent membership changes, the costs of rekeying the group can be quite high. Furthermore reliability is not guaranteed because the failure of the centralized controller is fatal to the whole system.

Chapter 6

State of the art

6.1 Logical Key Hierarchies

The use of logical key trees (hierarchies) for improving the scalability of group rekeying was proposed by Wallner [WHA99] and Wong [WGL00]. The basis for the LKH approach to scalable group rekeying is a logical key tree which is maintained by the key server (see fig. 6.1 [SZJ02]). The root of the key tree is the group key used for encryption/decryption of data and it is shared by all users. The leaf nodes of the key tree are keys shared only between the individual users and the key server, whereas the middle level keys are auxiliary KEKs used to facilitate the distribution of the root key.

In this scheme, each user owns all the keys on the path from its individual leaf node to the root of the key tree. As a result, when a user leaves the group, in order to maintain forward data confidentiality, all those keys have to be changed and redistributed.

There is a soft real-time requirement for key delivery. To address this issue, i.e. to reduce the latency of key delivery, group rekey transport protocols can make use of proactive redundancy.

Furthermore, the rekey payload has a sparseness property, i.e. while the packets containing the new keys are multicast to the entire group, each receiver only needs the subset of packets that contains the keys of interest to it. Thus, if a receiver-initiated protocol is used for reliable multicast, a receiver need only provide negative feedback for packets that contain keys of interest to it.

6.1.1 Periodic Batch Rekeying

Periodic batched group rekeying has been shown to reduce both the processing and the communication overhead at the key server, and to improve the scalability and performance of key management protocols based on logical key trees. Processing

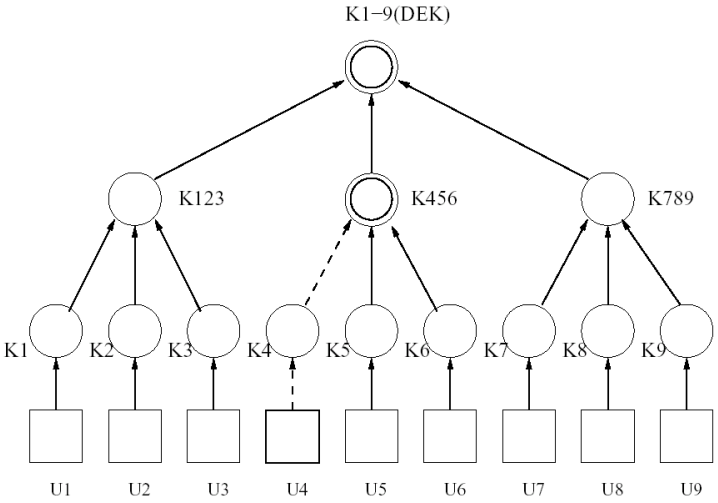

Figure 6.1: **Example logical key tree**

periodic group rekeying has the advantage of reducing the number of group rekey events in a given period of time at the expense of increasing the join and leave latency.

The reliable key delivery problem is particularly challenging when group rekeying is done periodically. In this situation, the number of keys that need to be changed can be large enough that a very large number of packets need to be multicast reliably to the whole group.

6.1.2 Wka-Bkr

Setia presented WKA-BKR [SZJ02], a scalable rekey transport protocol for group key management schemes that are based on logical key hierarchies. Its reliable key delivery protocol is based upon proactive redundancy and batched key retransmission. It uses proactive redundancy for achieving reliability and ensuring timely delivery of keys.

Setia's protocol uses a Weighted Key Assignment (WKA) algorithm that exploits the properties of a logical key hierarchy while assigning keys to packets that are multicast to the group. The keys at higher levels of the logical key tree are more valuable than other keys since they are needed by a larger fraction of the group's members, so by setting the degree of replication of each key based upon its position in the logical key tree, more valuable keys have a larger degree of replication.

WKA-BKR does not use FEC but batched key retransmission (BKR), i.e. instead of resending a lost rekey packet to the group, the protocol determines the keys that are needed by the receivers who responded with NACKs to the initial multicast, packs

these keys into new packets and multicasts them to the group. Interesting considerations for minimizing key lengths without losing security properties can be found in [PMM03].

6.2 The IETF Multicast Group Security Architecture

Hardjono and Weis [HW04] defined a Reference Framework for the elements of a secure multicast architecture. This framework defines the treatment of data sent to a group, the management of keying material used to protect the data, and the policies governing a group. Hardjono also defines the concept of Group Security Associations (GSA). In particular GSAKMP[HMCG04] is defined for group key generation and dissemination.

GSAKMP separates group security management functions and responsibilities into three major roles: 1) Group Owner, 2) Group Controller Key Server, and 3) Group Member.

The Group Owner is responsible for creating the security policy rules for a group and expressing these in the Policy Token.

The Group Controller Key Server (GC/KS) is responsible for creating and maintaining the keys and enforcing the group policy by granting access to potential Group Members (GM) in accordance with the Policy Token.

To enforce a group's policy the Group Members need to know who potentially will be in the group and need to verify that the key disseminator is authorized to act as such.

GSAKMP provides mechanisms for cryptographic group creation and management that may be used in conjunction with other protocols to allow various applications to create functional groups according to their application-specific requirements.

6.3 Source Authentication

The session key approach is inadequate for source authentication. Public key signature is the most natural mechanism for multicast authentication but has the disadvantage of being computationally expensive.

Canetti [CP99] presents solutions to the source authentication problem based on shared key mechanisms, where each member has a different set of keys. Main saving is in the signature generation time reduction.

In his solution each MAC is computed with a different key. Each recipient holds a subset of the keys and verifies the MAC according to the keys it holds. Appropriate choice of subsets ensures that with high probability no coalition of up to w colluding bad members (where w is a parameter) know all the keys held by a good member, thus authenticity is maintained.

A considerable gain in the computational overhead of the authentication scheme is achieved by noticing that the work needed for computing some known MAC functions on the same input and different keys is far less than the times the work to compute a single MAC. This is so since the message can first be hashed to a short string using key-less collision-resistant hashing. Using similar parameters to those of the basic scheme, one can guarantee that each good member has many keys that are known only to itself and to the sender. In order to break the scheme an adversary has to forge all the MACs computed with these keys. Thus it is enough that the sender attaches to the message only a single bit out of each generated MAC. Consequently, the total length of the tag attached to the message can be reduced to a few bits.

Naor [CGI+99] proposes a scheme enabling a single source to transmit to a dynamically changing subset of legitimate receivers from a larger group of users, such that coalitions of at most k users cannot decrypt the transmissions unless one of them is a legitimate receiver. The overhead of a rekeying message in this scheme does not depend on the number of users that are removed from the group. The communication overhead of the scheme is $O(k\ log^2\ k\ log(1/p))$, where p is an upper bound on the probability that a coalition of at most k users can decrypt a transmission to which it is not entitled. The scheme also requires each user to store $O(log\ k\ log(1/p))$ keys.

The main drawback of this scheme is that the security is only guaranteed against coalitions of up to k users, and the parameter k substantially affects the overhead of the scheme.

Chapter 7

Main Contribution

7.1 MTLS

In this section we describe a secured version of MFTP we propose, pursuing the following goals:

1. simplicity and effectiveness;
2. reduced performance loss over the non-secured version;
3. scalability;

Solutions proposed for unicast transmission, such as TLS [DR04], are not feasible for multicast, for both scalability and group dynamic management issues. We propose Multicast Transport Layer Security (MTLS) in [DGLS02a] to address the generic multicast security requirements. According to this proposal, session keys (NetKeys) are distributed by making use of an additional key, the key encryption key (KEK).

We develop a Java implementation of MTLS and run a series of tests to evaluate its impact on the performance of JMFTPv2, an improved (as regards packet send/receive performance) version of JMFTPv1 [BDGL01]. Whenever the version will be of no relevance, both JMFTPv1 and JMFTPv2 will be referred to simply as JMFTP in the following. More in detail, we evaluate: the influence of the encryption/decryption process on the maximum average throughput of JMFTPv2; the delay introduced by the key distribution process, how it scales with respect to the number of clients and its influence on the average throughput of JMFTPv2.

The MTLS protocol adopts an n-root leaf pairwise keys architecture [WHA99] to deliver the encryption keys, both the KEK and the NetKeys. This architecture has been chosen for the implementation of a first prototype. MTLS adopts the RSA algorithm for encrypting the KEKs, whereas the KEKs themselves and the NetKeys follow the DES specification.

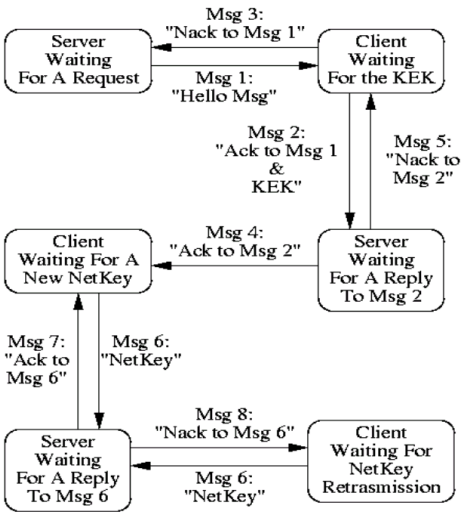

Figure 7.1: **MTLS state transition diagram**

The evolution of a single server-single client system adopting the MTLS protocol to safely distribute KEKs and NetKeys is described in figure 7.1. For the sake of simplicity the diagram does not take into account transitions due to time-out expirations. In other words we assume no message is lost by the network, nevertheless a message can be corrupted during its transmission: either by transmission errors, or by attempts of violating the security of the system.

The server can be in three different states. It can be waiting: for a request of the KEK (state ”Server Waiting for a Request” in the diagram); for a reply upon the transmission of the KEK (state ”Server Waiting For a Reply To Msg 2”); for a reply upon the transmission of the NetKey (state ”Server Waiting For A Reply To Msg 6”).

The client can be in three different states, waiting: for the KEK (state ”Client Waiting For the KEK”); for a NetKey not previously transmitted (state ”Client Waiting For a New NetKey”); for the retransmission of a Netkey, whenever it has already been received but resulted to be corrupted (state ”Client Waiting For a NetKey Retransmission”). Under the assumption that no message is lost by the network, state transitions are determined by the reception or the transmission of a message.

Exchanged messages are: Msg 1: ”Hello Msg”. The message by which a request for the KEK is submitted; Msg 2: ”ACK to Msg 1 and KEK”. The message, reports on the reception of a not corrupted request for the KEK and delivers the KEK to the client; Msg 3: ”Nack to Msg 1”. The message reports on the reception of a corrupted KEK request; Msg 4: ”ACK To Msg 2”. The message reports on the reception

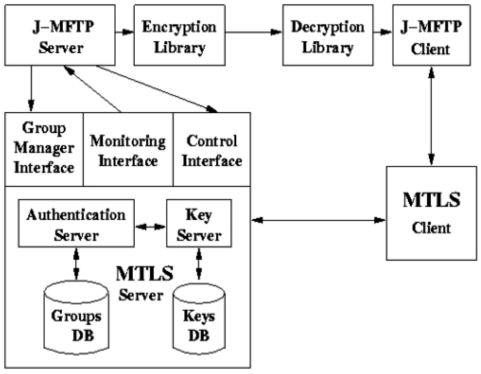

Figure 7.2: **Joint JMFTP-MTLS software architecture**

of a not corrupted KEK; Msg 5: "Nack To Msg 2". The message reports on the reception of a corrupted KEK; Msg 6: "NetKey". The message delivers to the client a NetKey; Msg 7: "ACK To Msg 6". The message reports on the reception of a not corrupted NetKey; Msg 8: "Nack To Msg 6". The message reports on the reception of a corrupted NetKey.

7.1.1 Implementation

The main components of the MTLS architecture are: a cipher library for data encryption; a decipher library for data decryption; the MTLS server and the MTLS client, the two components that can be integrated with any multicast application.

All the components of MTLS are implemented in Java, and are compliant to the Java Beans standard. The MTLS server and client are easily integrable with any Java multicast application, and fit as building blocks in the JMFTP framework.

The MTLS server has two main modules, the Key Server and the Authentication Server, and has a well-defined interface towards the application. The MTLS client has a similar interface towards the client application. An exchange message protocol has been defined in order to safely distribute NetKeys. The client authentication task is accomplished by the Authentication Server using security certificates. Furthermore the Authentication Server keeps track of client authorizations. The MTLS API is composed of the Group Manager Interface, the Monitoring Interface and the Control Interface. The Group Manager Interface provides services to add or remove users from the users list. The Monitoring Interface provides services to asynchronously notify the multicast application about relevant events concerning the distribution of NetKeys to authorized clients. The Control Interface provides services to notify the Key Server about a new session and to claim redistribution of the Netkey.

7.1.2 Details

MTLS provides both authentication and authorization in a multicast environment. Figure 7.2 shows a simplified view of the framework comprising JMFTP and MTLS and describes the points of integration between the two: JMFTPv2 server includes the proper sequence of MTLS method invocations; JMFTPv2 server interfaces with the encryption classes provided by MTLS; JMFTPv2 client includes the client side of the key distribution module provided by MTLS; JMFTPv2 client interfaces with the decryption classes provided by MTLS.

The dynamic behavior of the integrated JMFTP/MTLS system is described by the following sequence of actions:

1. the JMFTPv2 server shares the list of authorized users with the MTLS server, so that MTLS can authenticate the JMFTPv2 clients;

2. the JMFTPv2 server starts the announcement phase;

3. when a JMFTPv2 client joins the session, it requests the decryption NetKey to the MTLS Key Server via the MTLS client;

4. the MTLS Authentication Server authenticates the client and starts distributing the NetKey;

5. when the Netkey is successfully delivered to a client, the MTLS notifies the JMFTPv2 server;

6. when a proper number of clients, according to the policy implemented by the JMFTPv2 server, have joined the session and received the NetKey, file distribution starts;

7. during file distribution, the JMFTPv2 server can invoke the distribution of a new Netkey, either to periodically change the key or because an attack to the security system has been detected or is possible (e.g. when the group composition changes). It is worth noticing that we assume all the entities involved in the integrated system possess valid certificates.

7.2 JMFTP-MTLS Experimental Measurements

In this section we report on the performance of JMFTP and MTLS, with the goal of evaluating the impact of security on the overall performance of JMFTP [DGLS02b].

We are interested in measuring the following:

- the Host Transfer Time (HTT) in absence of rekeying (we expect that the encryption/decryption process has a measurable impact on this indicator);

- the Host Transfer Time (HTT) using scheduled rekeying;
- the experienced DTU loss rate;
- the rekeying time;

During a data distribution the NetKey should be changed at regular intervals to improve security. This process could have an influence on the HTT. During the distribution of the file, the NetKey could be suddenly changed since a security risk has been detected. The file transmission must then be suspended, the NetKey redistributed and the file transmission resumed. This may increase the HTT.

With the aim of evaluating these performance indicators, we first measure the achievements of JMFTP in generating possibly isochronous flow of packets corresponding to a nominal throughput. In our tests we adopt the same emission strategy presented in [BDGL01].

7.2.1 Evaluation

The strategy effectiveness is tested in a simple scenario in which a JMFTP server is connected to a JMFTP client through a 100Base T Ethernet segment. A third personal computer connected to the LAN segment runs a network monitoring tool. The JMFTP server and client are hosted on two Linux platforms: a Hewlett Packard pc, Xeon 550 MHz, 256 MB RAM and a Compaq PC, Xeon500 MHz, 128 MB RAM, respectively. Three different software target configurations are considered : JMFTPv1, JMFTPv2 with disabled cryptography, JMFTPv2 with enabled cryptography.

We measure the average throughput in transmitting a 10.0 MB file through MFTP packets of size 1.500 bytes. The results of our tests are shown in figure 7.3: both JMFTPv1 and JMFTPv2 perform well up to 6.0 Mb/sec (Mb = Megabits in this paper). For this emission rate, the actual throughput is 0.1 Mb/sec away from the nominal one in the worst case (JMFTPv2 with encryption). Furthermore, if the system transmits at the highest possible rate, JMFTPv2 with disabled encryption, achieved a remarkable 25.5 Mb/sec, whereas JMFTPv1 doesn't exceed 9.6 Mb/sec. Encryption has a strong influence on the performance of JMFTPv2 server, whose emission rate in this case does not exceed 6.45 Mb/s.

In the following, we report the results of tests on MTLS by itself and in conjunction with JMFTPv2. In these tests we consider: the time required to deliver the NetKey to a single client; the time required to deliver the NetKey to a growing number of clients; a sketch of a scalability test; an evaluation of the performance degradation induced by the concurrent delivery of both NetKeys and JMFTP packets.

Figure 7.4 shows the relation existing between the emission throughput and the average values measured for the HTT. It is worth noticing that, in the considered scenarios, with a high probability, packets can only be lost by the clients because of

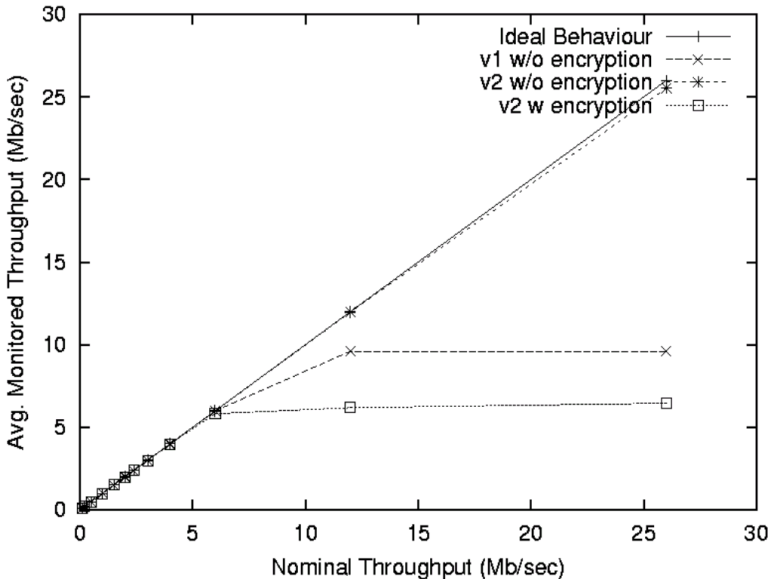

Figure 7.3: **Nominal vs. Monitored throughput**

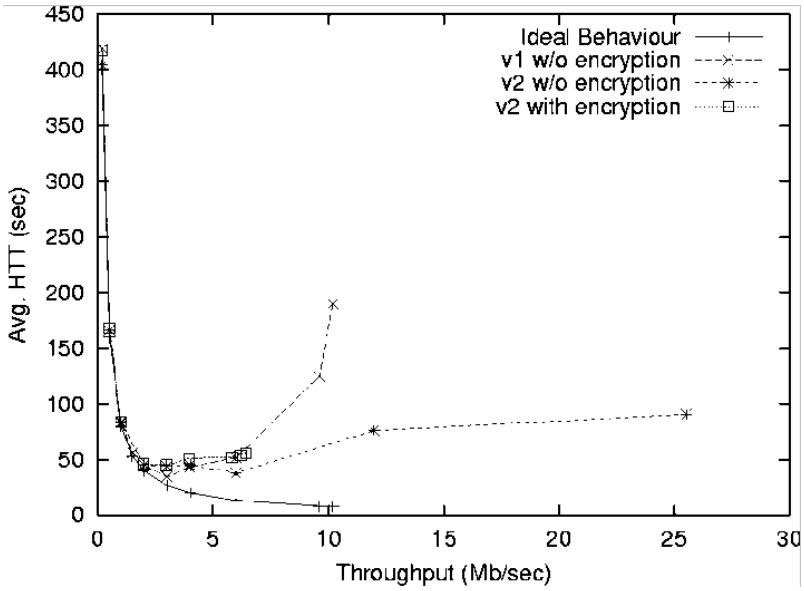

Figure 7.4: **HTT dependence on the emission throughput**

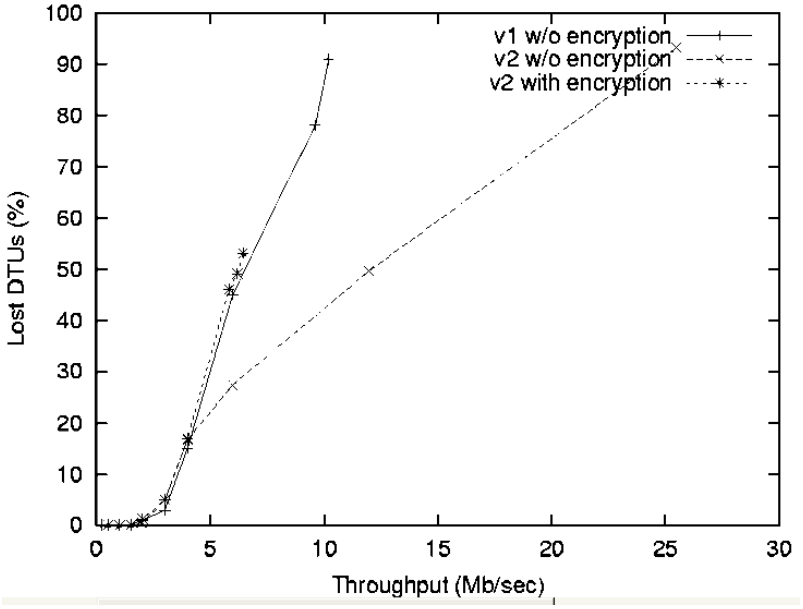

Figure 7.5: **Relation between emission throughput and DTU loss**

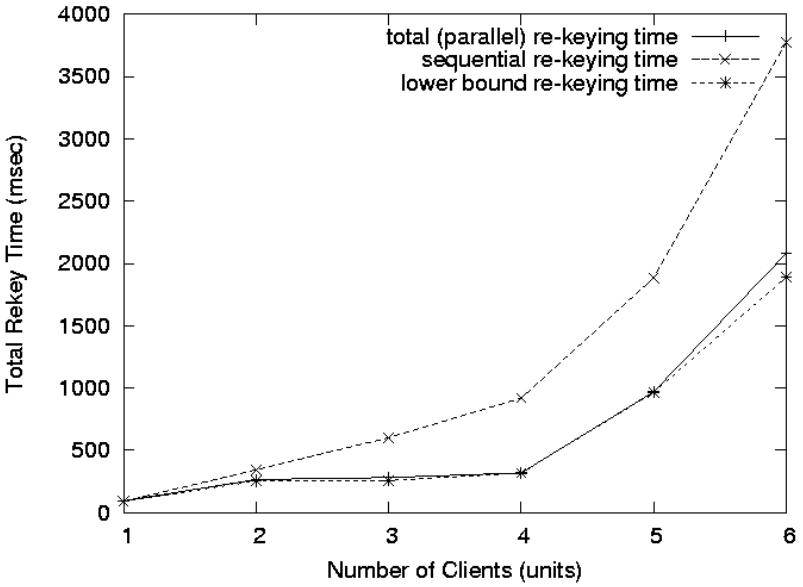

Figure 7.6: **Total Rekeying Time for variable number of heterogeneous clients**

Host	First (ms)	Subsequent (ms)
Compaq Deskpro	206	90
Toshiba Tecra	440	254
Pentium III	399	258
Sun Ultra 5	496	317
Texas Extensa	2998	965
Acer Travelmate	2100	1890

Table 7.1: Rekeying times

their inability to sustain the transmission throughput. In addition the best system performance is not obtained at the highest achievable transmission rate. The minimum HTT is reached when the emission throughput is roughly between 3.0 and 6.0 Mb/sec for all considered target software configurations: 6.0 Mb/sec for JMFTPv2 with encryption disabled; 3.0 Mb/sec for JMFTPv2 with encryption enabled and JMFTPv1.

Figure 7.5 shows the relation existing between emission throughput and DTU loss rate. It can be observed that: JMFTPv2 client performs better than JMFTPv1 client in that the probability of packet loss induced by the client itself is significantly lower; encryption/decryption has a strong influence on the client performance when the throughput exceeds 4.0 Mb/sec.

Table 7.1 shows the rekeying times referred to different hardware/software client configurations in a one-to-one scenario. We can observe that the time required to deliver the first NetKey is greater than that required for rekeying. We argue this is due to the time spent in accessing the disk and loading the necessary Java classes.

Figure 7.6 shows the measured total rekeying time (TRT) as a function of the number n of (heterogeneous) MTLS clients involved. The TRT is defined as the time elapsed from the invocation of the rekeying procedure to the reception of the last NetKey acknowledgment. The experiment was performed as follows. We had a set of available rekeying client machines, each representative of a performance class (see table 7.1). The test for $n = 1$ involved the fastest client. The test for $n = 2$ involved the two fastest clients. The test for $n = 3$ the three fastest clients, and so on.

The parallel rekeying process MTLS performs allows to obtain the performance shown in the curve labeled as total (parallel) rekeying time in figure 7.6. In the same figure the sequential rekeying time is shown, given by the sum of the one-to-one rekeying times of the clients involved in the test, as reported in table 7.1. The lower bound curve plots the one-to-one rekeying time of the slowest client participating to the rekeying procedure, which is the obvious performance limit.

The efficiency of the implemented MTLS parallel rekeying strategy is clear, because the lower bound and total parallel rekeying time curves are almost identical. As a consequence we can foresee that the behavior of the rekeying operation for larger

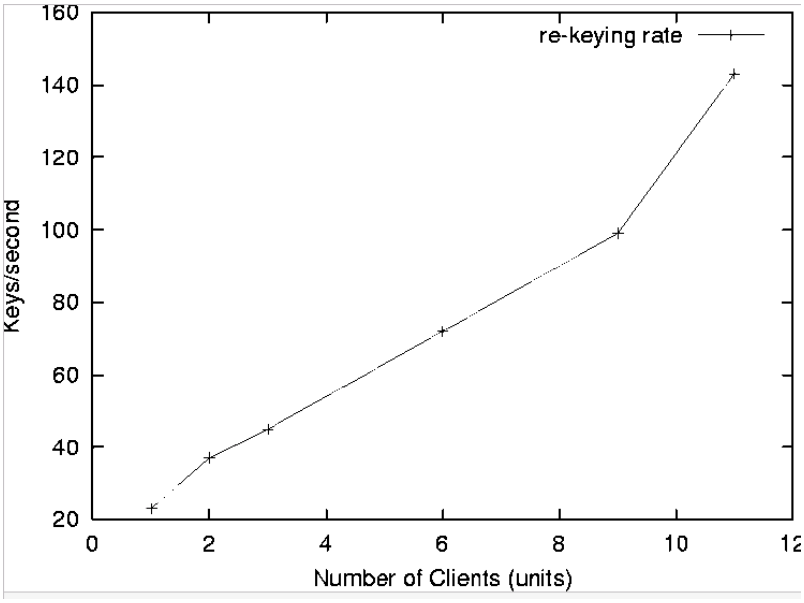

Figure 7.7: **Keys per second with respect to the number of clients**

groups will be limited mostly by the performance of the slowest class of clients.

On the basis of the results we collected, it is possible to conclude that: JMFTPv2 substantially performs better than JMFTPv1, especially for emission throughputs exceeding 4.0 Mb/sec; the impact of encryption/decryption on the overall system performance (min HTT) is negligible and the system performance when encryption is disabled is pretty good up to high emission throughputs.

In other words the MTLS server can properly and concurrently serve multiple requests at the rate it was solicited during this experiment. We have made a preliminary evaluation of the correctness of the system for sessions with tens of clients, with satisfactory results.

Another performance indicator for the rekeying system is the number of NetKeys the server can produce in a second. A relevant fraction of the time required to generate a single NetKey is spent for its encryption, using a different key for each client. These tests are performed by installing several MTLS clients on particularly fast machines in order to stress the server. Figure 7.7 shows the maximum measured throughput corresponding to a given number of clients in the session.

By examining the results of the tests concerning MTLS we can claim that the first rekeying operation has an appreciable overhead with respect to the following ones; due to the use of Java threads in the Key Server implementation the total rekeying time could moderately scale with respect to the number of clients.

Finally, we can make some considerations on the behavior of the integrated sys-

tem: the impact of the scheduled rekeying process onto JMFTP performance is negligible, when concurrently performed with data transmission; however, in the case of a sudden rekeying, the data transmission process has to wait as long as the new NetKeys have been distributed to all participants. This could result in a significant delay on the total and on the host transmission times, impacting performance; The measured influence of the data transmission over the rekeying time of a target client can roughly be estimated as equal to 10%.

We can argue that, in our tests, the impact of encryption/decryption on the global system performance is negligible when the emission rate is moderately high (i.e. when it does not exceed the value of 4.0 Mb/sec).

Starting from an effective throughput of 4.0 Mb/sec, the decryption process has an increasing role in slowing down the JMFTP client, and as a consequence in the packet loss. Encryption starts impacting system performance from an emission rate of 6.0 Mb/sec, that is the maximum throughput achievable by the server itself.

The time needed to deliver one NetKey to a single client is strongly dependent on its hardware and software configuration. During our tests, the MTLS server effectively used the multithreading facilities, but we didn't succeed in reaching the performance limit of the server itself. The total rekeying time is negligible in the case of a scheduled request of rekeying. However, in the case of a sudden rekeying, a significant and critical delay on the total and host transmission times is possibly introduced.

The overall performance of the secured system is still not adequate to high performance environments. Rekeying scalability needs improvements. Results show that the bottlenecks of the systems are at the client side of both software components: the performance of the MTLS client must be in-depth investigated to speed-up its response to the server; the sustainable throughput of the JMFTP client can be improved further.

7.3 Concluding Remarks

In this part of the thesis we have briefly exposed a survey of secure multicast issues, problems and solutions. We have presented some mainstream proposals on how to handle key management in multicast environments. Not all the possible approaches were presented here but some of the alternatives that represent the current directions in this area.

The main problem in multicast key management is scalability. The research in this area has been going on for some years now and there is not yet a clear consensus on how the keys should be distributed. IETF is also working in this area, and that will help produce standardized and more widely adoptable solutions.

We have gained experience by extending our reliable multicast framework with security modules and experimenting with a reliable multicast protocol and key distri-

bution algorithm.

We have contributed with a reference reliable secure multicast protocol and a real-world performance analysis. Trade-offs between security and performance have been found, while our research effort is continuing, further results are being collected and more scalable solutions to the key distribution problem are being studied.

Part III

Reliable Multicast Applications

Chapter 8

Replica Management and Grid

8.1 Introduction

Very often, users of scientific applications require remote access to very large datasets. A technique for access latency reduction and efficient network resource usage can be to replicate frequently accessed datasets at locations near the eventual users. Such replication has to be performed in a reliable and efficient way in order to be convenient.

In scientific Grid applications, data volumes generated and collected at different sites can reach the order of petabytes [Cas02]. Large Grid applications are related to High Energy Physics, Earth Observation and Bioinformatics [HJS+00]. HEP is a large collaboration where over 2,000 researchers distributed all over the world analyze the data generated by a single accelerator. For Earth observation, data are collected at distributed stations and are maintained in geographically distributed sites. In molecular biology and genetics research a large number of independent databases need to be integrated in a single logical one. The users of such applications require a ubiquitous fast read/write access to up-to-date data, independently from the site where data originated.

In this chapter we will survey issues including:

- Data replication strategies;
- Grid technologies;
- (Data) replica management systems.

8.2 Replication Strategies

Data replication can improve system scalability, fault tolerance and load balancing, and as a consequence increase access performance and guarantee a better data

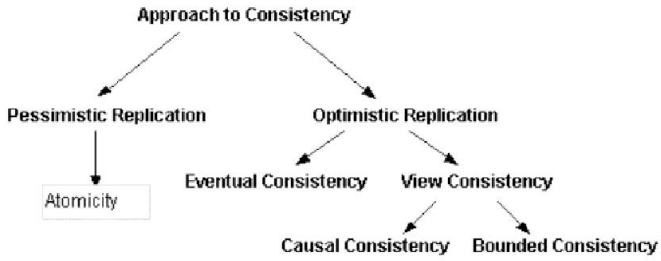

Figure 8.1: Approaches to consistency

availability. It mainly consists of maintaining a number of copies, referred to as replicas in what follows, of the same data items on different (geographically distributed) sites [GKL+02]. Replicas must be kept consistent and up-to-date. As a consequence writes and reads can be more complex and possibly slower.

As shown by figure 8.1, two different approaches are adopted by Replica Management Systems in order to deal with inconsistencies among replicas: the *eager*(pessimistic, synchronous) and *lazy*(optimistic, asynchronous) approaches [SL00].

According to the first one, replicas are kept synchronized by updating all of them as part of a single global atomic transaction, thus a priori excluding inconsistencies. Unfortunately, the higher the number of replicas, the longer the time spent in the distributed locking process. This is one of the main reasons why the *eager* approach does not scale. Furthermore, it imposes strong resource requirements and heavy limitations on replica availability [BHG87]. Finally, implementing updates as part of a single global transaction is not suitable for scenarios where sites are often disconnected from the network, as in mobile network environments [JPOD96].

According to the *lazy* approach, updates are asynchronously propagated to all other sites after the local transaction has been committed. Most crucial issues of the *lazy* solution are replica consistency management and performance scaling. *Lazy* replication cannot ensure a strict synchronization by definition, thus it is important to define which level of consistency the system supports.

Two main levels of consistency are possible: *eventual* and *view*.

Eventual consistency guarantees that if no new update submission occurs and sites can freely and reliably communicate for an unbounded interval of time, all replicas will eventually converge to the same content. This minimal requisite is suitable for most applications.

Whenever more stringent requirements have to be met, *view* consistency guarantees an active control on the quality of data by regulating read and update submissions. A view is consistent with the actions of an entity if it provides a data version that is not older than what an entity has seen previously. However, this approach requires tracking the last version of every file that has been accessed by an entity or user in the past, hence the view consistency database can become very large.

View consistency includes *causal* consistency that preserves partial orderings among read and write requests, and *bounded* inconsistency that explicitly limits the degree of replica inconsistency.

Temporal consistency is a particular case of bounded consistency guaranteeing that replicas will converge to the same content within a prefixed interval of time, by letting newest object contents propagate to the replicas within a fixed period. Data temporal consistency constrains how old a data item may be and still be considered valid [CPW97]. This latter is an effective and less resource-hungry consistency requirement.

Finally, it has also to be mentioned that whereas *eager* replication delays or aborts a transaction when committing would violate serialization [JPOD96], *lazy* replication has a more difficult task, since some replica updates could have already been committed when a serialization problem is detected. Thus, in case of conflict, transactions have to be reconciled by the system.

Three phases are needed for *lazy* replication in order to reach consistency:

1. Updates propagation;

2. Updates scheduling;

3. Detection and reconciliation of conflicts among updates;

In the next sections we will describe a real-world application scenario for the replica management problem, analyze it more in depth and propose a general solution for the above-mentioned first phase.

8.3 The Grid

Available computational resources today do not always meet the demand of increasingly complex scientific applications. Improvements in wide-area networking make it possible to aggregate distributed resources in various collaborating remote institutions. The main result of such aggregation today is known as Grids.

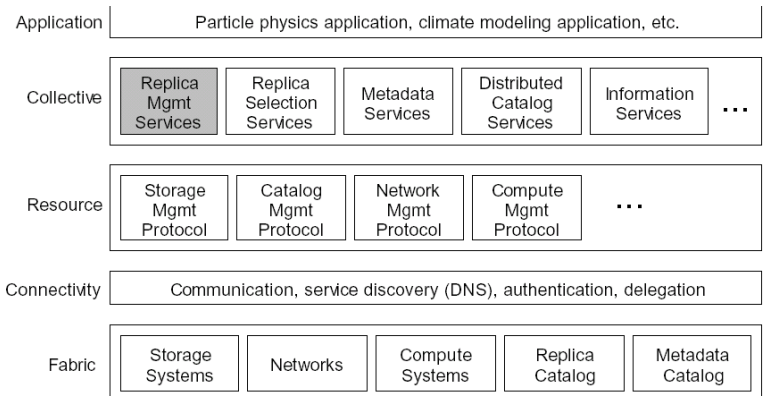

Figure 8.2: **Components in a layered Grid architecture**

On the one hand, computational Grids main purpose is to offer power for computationally intensive applications that need fast elaboration on (relatively) small data sets. Data Grids, on the other hand, are created to support applications that operate on very large data sets (oftentimes in the order of petabytes). In this context, data replication management is a particularly important issue. In the following, when talking about Grids, we will mainly refer to data Grids and in particular to the European DataGrid project[Dat], both a computational and data-intensive Grid that has been adopted by our research group. It is worth mentioning the Global Grid Forum [For] contribution to the development of Grid technologies, via the elaboration and dissemination of ”best practices”, implementation guidelines and standards.

In ”The Anatomy of the Grid” [FKT01], Foster and Kesselman define the Grid problem as ”coordinated resource sharing and problem solving in dynamic, multi-institutional, virtual organizations”. They also define the architecture whose layers are depicted in fig 8.2 [All]. Figure 8.2 shows some components of the Data Grid reference architecture that are relevant to replica management.

The software infrastructure presented in that paper and in [FKNT02] places a large emphasis on *interoperability* as it is fundamental to ensure that *virtual organization*(VO) participants can share resources dynamically and across different platforms, programming environments and languages. The importance of Foster’s architecture resides mainly in its being a reference framework where many researchers and institutions have reliably been working for years. Other contributions to Grid computing have been proposed, all of them had to refer to Foster’s work.

Figure 8.3 shows a complex Grid reference model.

Figure 8.4 shows the DataGrid architecture we refer to. Each box represent a Grid element, i.e. a server which provides a particular functionality within the Grid. We briefly describe the role of each machine:

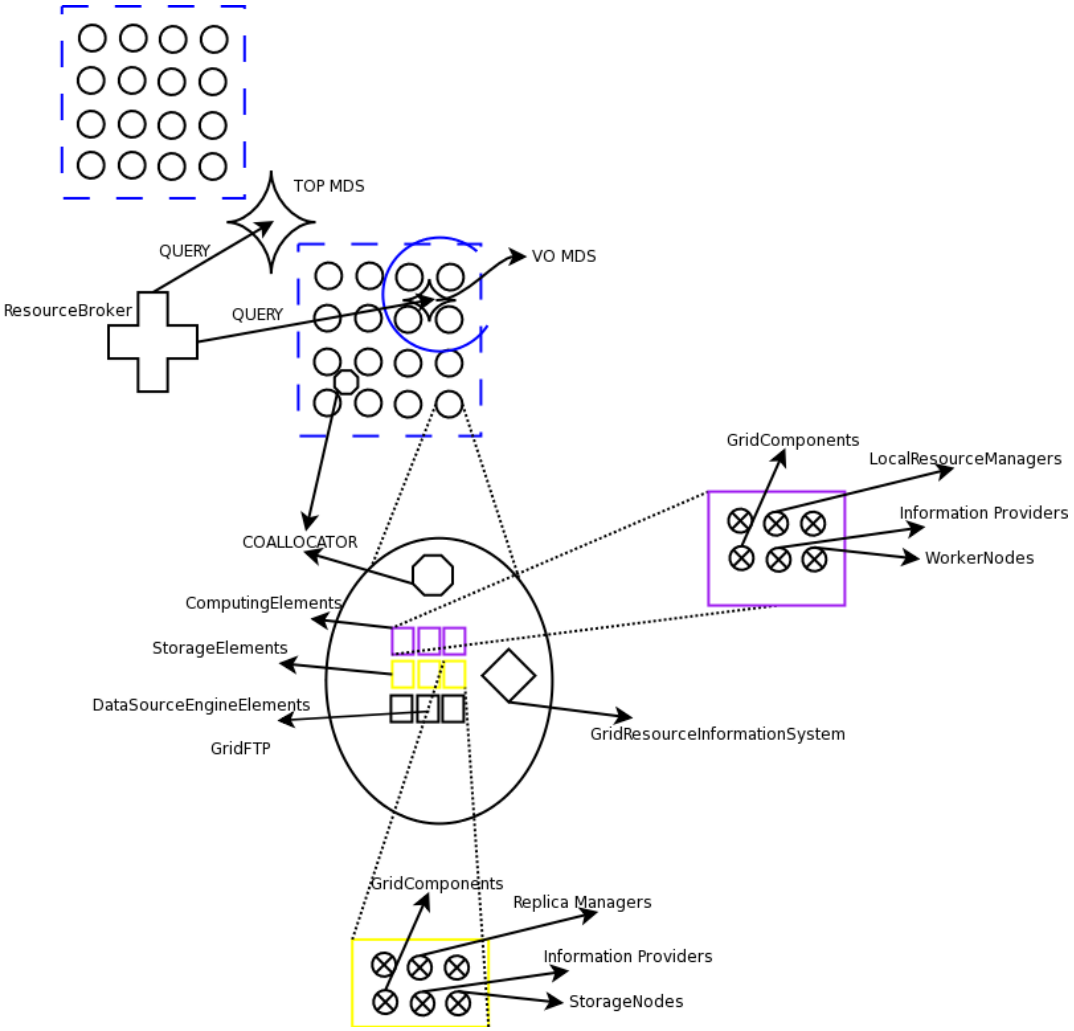

Figure 8.3: **A Grid reference model**

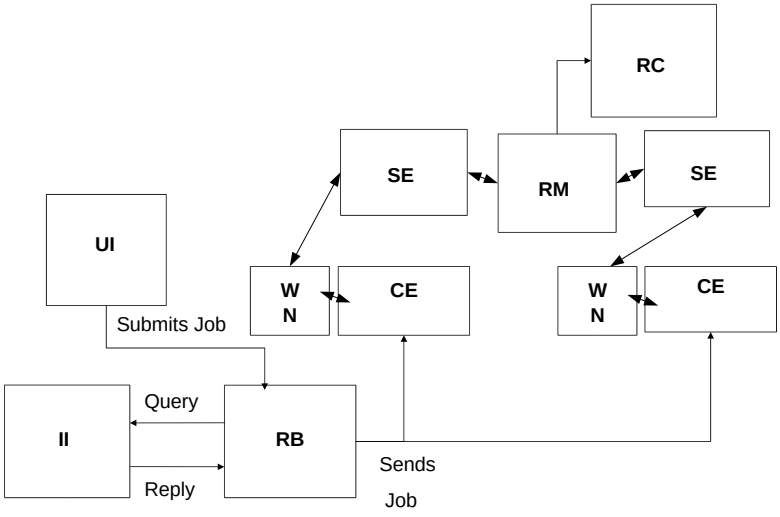

Figure 8.4: **Elements in a DataGrid**

- The User Interface (**UI**) is the machine that provides user access to the DataGrid services. The user interacts with the Grid via command line or using more advanced interfaces.

- The Resource Broker (**RB**) is the server that receives users' requests from the UI and queries the Information Index to find suitable resources (ex.: a computing element which satisfies user's requirements).

- The Information Index (**II**) maintains information about the available resources.

- The Replica Manager (**RM**) is used to coordinate file replication from one Storage Element to another. This is useful for data redundancy but also to move data closer to the machines which will perform computation.

- The Replica Catalog (**RC**), manages the information about file replicas. A logical file can be associated to one or more physical files which are replicas of the same data. Thus a logical file name can refer to one or more physical file names.

- The Computing Element (**CE**) is the server which receives job requests and delivers them to the Worker Nodes. The Computing Element provides an interface to local batch queuing systems (e.g. PBS, LSF,...).

- The Worker Node (**WN**) processes input data and produces output for a job.
- The Storage Element (**SE**) is the system providing storage space. It provides a uniform interface to different Storage Systems.

8.4 The Replica Management in the Data Grid

In these sections we focus on the problem of data replication in the Data Grid.

Replicas here are defined in terms of files, not objects, but there exist well-defined mechanisms for mapping objects to files in a way that can be completely transparent to applications.

The use of replicas is transparent to users; they are created as needed by the Grid middleware in order to improve overall performance of jobs [BCC+02] [CCSM+03]. However, sites can explicitly ask for the creation of local replicas.

The Replica Manager keeps track of all replica data so that the replica selection service can select the optimal physical file to use for a given job, or to request the creation of a new replica.

The main Grid terms for replica management are:

Logical File Name (LFN): a logical file is identified by a globally unique string that is independent from where a file is physically stored, and from the protocols that can be used to access it.

Physical File Name (PFN): a physical file name is used to uniquely identify a file on a given SE.

Transport File Name (TFN): a transport file name is used to identify how a file is to be accessed. The TFN allows multiple protocols to be used to access a single PFN (e.g. GridFTP, ftp, http,...).

8.5 Globus

Large scientific applications on the Grid have three main requirements on data management in common, such as:

- A secure, reliable, wide-area data transfer protocol.
- Services for registering and locating files.
- The management of multiple copies of files.

Grid components satisfying these requirements can be found in Globus. The Globus project provides middleware services for Grid computing environments.

Main Globus components are:

1. The Grid Security Infrastructure (GSI) provides authentication and authorization services using public key certificates and Kerberos authentication.

2. The Resource Allocation Manager(GRAM) architecture provides a way for specifying application requirements and (GARA) mechanisms for resource reservation.

3. The Information Management architecture (MDS) provides a scheme for publishing and retrieving information about resources in the wide area environment.

4. The Data Management architecture, providing:

   - GridFTP, a universal data transfer protocol for grid computing environments.
   - Global Access to secondary storage (GASS)
   - A Data Catalog and Replica Management Infrastructure for managing multiple copies of shared data sets.

In different Grids often heterogeneous application-specific storage systems are used, such as the Distributed Parallel Storage System (DPSS), the High Performance Storage System (HPSS) and the Distributed File System (DFS). Unfortunately, most of them utilize incompatible protocols for accessing data thus effectively partitioning the datasets available on the Grid.

The GridFTP protocol can be used to overcome these incompatibilities by providing data movement in Grid environments.

The Globus Replica Management Infrastructure is responsible for managing complete and partial copies of data sets. Services provided by a replica management system include:

- Creating new copies of a complete or partial data set;
- Registering copies in a Replica Catalog;
- Allowing users and applications to query the catalog to find all existing copies of a particular file or collection of files;
- Selecting the best replica for access based on storage and network performance.

8.5.1 GridFTP

GridFTP is the Globus data transfer and access protocol that provides secure data movement in Grid environments [All01]. This protocol extends the standard FTP protocol and includes the following features:

- GSI and Kerberos support;

- Third-party control of data transfer: GridFTP allows a user or application at one site to initiate, monitor and control a data transfer operation between two other sites;
- Parallel data transfer: GridFTP supports parallel data transfer using multiple TCP streams in parallel to improve aggregate bandwidth over using a single TCP.
- Striped data transfer: Data may be striped or interleaved across multiple servers, which use multiple TCP streams to transfer data that is partitioned to provide bandwidth improvements.
- Partial file transfer: GridFTP provides commands to support transfers of arbitrary subsets or regions of a file.
- Automatic negotiation of TCP buffer/window sizes;
- Support for reliable and restartable data transfer.

These improvements over standard ftp are needed in order to cope with the requirements of the large scientific Grid applications, i.e. very large datasets and the very large bandwidth (in the order of Gigabit/s) often available.

GridFTP is mainly focused on point-to-point file transfer, while it is not optimized for parallel point-to-multipoint transfer as it is required in replication. A proposal for improving the performance in such cases in given in [IGT04], where pipelining and multiple distribution trees decrease distribution time to multiple sites.

8.6 Replica Management Systems

Various Grid Replica Management Systems have been proposed to date. *GDMP* [SSMD02] is a file replication tool for Data Grid environments with support for automatic replication through a publish-subscribe notification system. GDMP uses GridFTP and GSI security and manages replica catalog entries for file replicas, thus maintaining a consistent view on locations of replicated files.

The *Edg-Replica-Manager* is the Globus Replica Management Infrastructure library, with adaptation to the EU DataGrid environment [Dat].

Reptor [KLSS04] is a replica management service prototype providing a programmer-friendly intuitive interface, hiding the details of the underlying services in a DataGrid environment. It is the successor to the Edg-Replica-Manager. Reptor provides replica management and optimization, whose goal is to select the best replica with respect to network and storage access latencies. The Reptor optimization service determines the replica that should be accessed from a given location and can potentially determine the best location for new replicas. Reptor supports the Web Services paradigm [Kre01] and can rely also on standard mechanisms (e.g. GridFTP) for file transport.

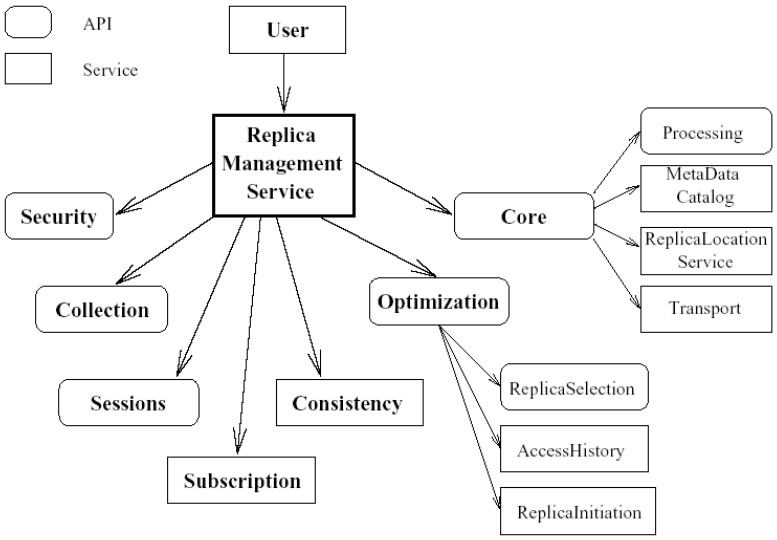

Figure 8.5: **Reptor logical layout**

It is worth noticing that even though Reptor offers an optimized replica management service through an easy-to-use interface, in reality it is a complex system, composed of a large number of interacting components, as shown in figure 8.5 [KLSS04].

8.7 Grid and Web Services

One of the key objectives for Grids is *interoperability* i.e. the ability of components developed by different vendors using different tools to work together. This is one of the main reasons why the Web Service approach is so interesting for the Grid community.

The Open Grid Services Architecture (OGSA) Framework [All] was proposed in 2002 by Globus and IBM for integrating the Web Service paradigm inside the Grid architecture and to standardize virtually all the services of a Grid application (e.g. job and resource management, security). OGSA defined an extended type of Web Service called Grid Service, i.e. a Web Service with extensions that make it suitable for a grid-based application.

The Open Grid Services Infrastructure (OGSI) was created with the objective that it would eventually converge with Web Services standards. Unfortunately, Grid Services have several drawbacks:

1. OGSI does not work well with current Web Services tools.

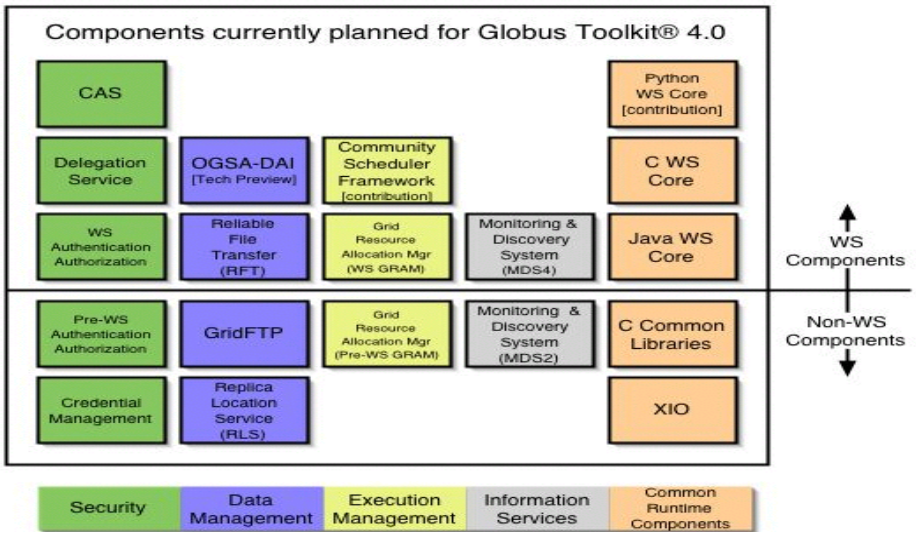

Figure 8.6: **The Globus Toolkit 4 architecture**

2. OGSI adopts a lot of OO concepts (statefulness, the factory/instance model,...) that are useless in OGSA.

that made this convergence infeasible.

The Web Services Resource Framework (WSRF) [All] standard was proposed to substitute OGSI. WSRF aims to integrate deeply with Web Services standards. OGSA will be based directly on Web Services instead of being based on OGSI Grid Services.

The Web Service approach is so promising that the new Globus Tookit 4 [All] (see fig. 8.6) will be the first available WSRF implementation.

Our research efforts already comprise Web Services, as we shall see in the last section of this part of the thesis.

Chapter 9

Main Contribution

9.1 Introduction

In this chapter, we describe our proposal for the problem of update propagation in multi-master replica management systems (see figure 9.1 [SL00]), i.e. systems where sites are allowed to update the replicas they manage independently from other copies, provided the changes will be subsequently propagated to all other sites. Our proposal applies multicast transmission to both LANs and WANs in order to support heterogeneous networks in a scalable way. By our approach, updates are lazy-propagated through reliable multicast, while replica state estimation is accomplished by a variant of the Timestamp Matrix technique. Each site multicasts its Timestamp Vector to all other sites so that they can maintain their own Timestamp Matrix.

The Timestamp Matrix makes it possible for a site to decide whether to reliably multicast the updates it is in possession of, or not, thus exploiting epidemic propagation of updates [RGK96], extremely useful in case of network partitioning.

When compared to previous approaches, our solution potentially reduces the amount of transmitted *log update* information, decreases propagation time and increases system scalability with respect to the number of sites. Our model captures the requirements of data-intensive replica services in scalable Grid environments that require systems to exchange update files in order to reach global consistency.

A Java prototype of our Replica Update Propagation System, named Gedec, has been implemented and functionality tests performed show the effectiveness of our approach. Gedec is the acronym for Grid-enabled distributed eventual consistency, which is the historical name of this research project.

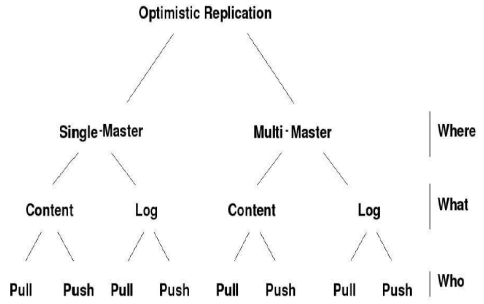

Figure 9.1: **Classification of replication models**

9.2 Gedec

9.2.1 Our Model

Let $S_1, S_2, \ldots S_n$ be n sites, connected by possibly heterogeneous networks (WANs as well as LANs). Let us simplify notation by saying each S_i maintains a replica R_i, also referred to as *object* in what follows, of the same data item. Each R_i can be independently updated (i.e. inserted, modified, deleted), provided the changes are eventually propagated to all other sites S_j, $j \neq i$ [AAS97]. In what follows we will assume that:

- sites share a common clock (but are not necessarily synchronized);
- sites share a common propagation scheduling time $t_0, t_1, t_2, \ldots$. For the sake of simplicity $t_0, t_1, t_2, \ldots = 0, 1, 2, \ldots$;
- time intervals between two propagation phases, i.e. $t_{i+1} - t_i$, $i = 1, 2, 3, \ldots$ are long enough to make it possible for each site to receive and process the updates propagated by all other sites.

Let us now consider an ideal scenario in which transmissions never fail and all sites are permanently connected to the network. Each site S_i maintains a set of records, referred to as log-file LF_i [RGK96] in what follows. Each record contains the information related to one single update operation, i.e. a description of the update operation, the time at which it occurred and the site at which it originated. Each time an update u is submitted at S_i, u is added to a log-file LF_i. Let t_p immediately follow t_{p-1} in the common propagation scheduling. In order to achieve global consistency and up-to-dateness, at time t, each site S_i propagates its log-file update $LFU_i(t_p)$ (comprising all updates submitted at site S_i between t_p and t_{p-1}) to all other sites.

Upon the reception by S_j of $LFU_i(t_p)$, its content is *processed* and R_j accordingly updated. The time elapsing between two subsequent log-file update propagation phases heavily affects the trade-off between consistency and up-to-dateness.

We consider now a more realistic scenario in which log-files updates could be lost and sites could be temporarily disconnected. Two orders of problems arise with respect to the ideal condition just considered:

1. sites are not aware of the log-file updates they have not received;
2. log-file updates of the disconnected sites are temporarily unavailable.

Various solutions to these problems have been proposed based on the selective retransmission of log-files updates. Basic elements of these solutions are:

- a Timestamp, intended as any number that increases monotonically, a logical clock, wall clock or counter [Lam78], is associated to each log-file LF_i.
- each site S_i maintains a Timestamp Vector TV_i of size n such that $TV_i[i]$ is the time stamp of the last log-file update generated at S_i and $TV_i[j] i \neq j$, is the timestamp of the last log-file update produced at S_j and processed by S_i.

In order to reach global consistency efficiently, sites have to exchange *status* as well as *log update* information. In some state of the art solutions, each site propagates its TV, and both its log-file update and a selection of the log-files updates generated by other sites. The idea here is that of epidemic propagation [RGK96], i.e. that of making it possible for a site to transmit all the information it is aware of, thus exploiting the knowledge about data location redundancy for improving the whole system robustness and shortening the time spent in achieving replica consistency. Each time S_i generates a $LFU_i(t_p)$, $TV_i[i]$ is correspondingly updated.

In order to propagate updates from S_i to S_j, S_j sends its TV_j to S_i. Upon the reception of TV_j, S_i sends S_j all $LFU_k(t_p)$, $k = 1,2,\ldots,n$ such that $TV_i[k] \geq t_p > TV_j[k]$. For each received $LFU_i(t_p)$ TV_j is updated if and only if $t_{p-1} = TV_j[i]$, otherwise $LFU_i[t_p]$ is stored for later processing.

TV_j is updated in the following way:

1. $TV_j[k]$ is set to the maximum between $TV_i[k]$ and $TV_j[k]$ for $k = 1,2,\ldots,n$, $k \neq i$;
2. $TV_j[i]$ is set to t_p.

Clearly, in order to implement epidemic propagation, $LFU_i(t)$s are not discarded immediately after they have been processed [WB84]. The main drawback of this *pull* approach is that any site can potentially receive the same log-file update more than once.

Another solution to the replica management problem is given by letting each site estimate the *progress* of other sites in achieving replica consistency and *pushing* only those *LFU*s that are likely to be missing at a remote site.

This estimation can be supported by the adoption of Timestamp Matrices [AAS97]. Each S_i maintains a n by n timestamp matrix TM_i such that:

- $TM_i[i][i]$ is the timestamp of the last log-file update generated by S_i;
- $TM_i[i][j]$, $j \neq i$, is set to t_p, where t_p is the most recent timestamp such that $LFU_j(t_0)$, $LFU_j(t_1)$, ..., $LFU_j(t_p)$, all have been received and processed by S_i;
- $TM_i[j][k]$, $j \neq i$, $k \neq i$ is a lower estimate of $TM_j[j][k]$.

Sites exchange their timestamp matrices, possibly piggybacking them on packet reception acknowledgments. Upon the reception of TM_i, S_j updates $TM_j[k][l]$, $k,l = 1,2,\ldots,n$, with the maximum between $TM_j[k][l]$ and $TM_i[k][l]$. Let us consider the generic l^{th}: whenever $TM_j[j][l]$ is the maximum among the timestamps in column l, S_j sends $LFU_j(t)$, with $t = TM_j[k][l]+1,\ldots,TM_j[j][l]$, to S_k, for all $k \neq j$ different from n.

Epidemic propagation is intrinsic to this use of timestamp matrices and is accomplished in a way similar to the timestamp vector case, the main difference being that sender site S_j relies on TM_j to estimate TV_i rather than receiving TV_i from S_i itself.

Finally, it is worth noting that using timestamp matrices this way does not solve the problem of duplicate updates.

9.2.2 Details

According to our proposal, updates(LFUs) are lazily propagated through reliable multicast, while replica state estimation is accomplished by an original variant of the Timestamp Matrix technique approach [WB84], based on multicast transmission of both *status* (TVs) and *log update* information (LFUs).

Each site multicasts its status information (TV) to all other sites so that they can maintain their own Timestamp Matrix (TM). The TM makes it possible for any site to decide whether to reliably multicast to other replicas the updates it is aware of, or not, thus exploiting epidemic propagation [RGK96].

The approach is characterized by the following:

- Each site maintains a timestamp matrix;
- Status information is multicast to all sites other than the originating one;
- Propagated status information is limited to a single timestamp matrix row(i.e. a TV): in particular each site S_i propagates the i^{th} row of TM_i to all other sites;
- Log update information is epidemically propagated through reliable multicast.

Furthermore, considering the inherently push nature of multicast transmission, this direction of transfer has been adopted (See Fig.9.1 [SL00]). Multicast transmission to a large set of receivers can theoretically offer the following advantages:

- Shorter total transfer times (i.e. faster distribution of the same file to all receivers);
- More efficient usage of network resources;
- Increased scalability with respect to the number of sites;

9.2.3 The Propagation Strategy

As already stated, both *status* and *log update* make use of multicast transmission. It is worth noting that only LFUs are epidemically propagated, i.e. any site is allowed to propagate LFUs generated at any other site. The reason why is that epidemically transmitting TVs can cause a delay in updating TMs, which can lead to inaccurate global system *status* estimation and possibly useless retransmissions.

As concerns the propagation of *log update* information our algorithm works as follows: let $M_{i,j} = \max\{TM_i[k][j], k = 1,2,\ldots,n\}$ and $m_{i,j} = \min\{TM_i[k][j], k = 1,2,\ldots,n\}$. If $TM_i[i][j] = M_{i,j}$, S_i starts a log sender election phase, which will be described in subsection 9.2.4, possibly propagating those LFUs generated by S_j, S_i is aware of, whose timestamp t is such that: $m_{i,j} < t \leq M_{i,j}$, for all $j = 1,2,\ldots,n$ [1].

It is worth noting how in this way it is possible for S_i to propagate LFUs to some other site S_k generated by some now disconnected site S_j. As an example S_k could not have received those LFUs since in turn he was disconnected at the time those LFUs had been propagated.

Concerning the transmission of *status* information:

1. Each site S_i multicasts its TV_i, i.e. the i^{th} row of TM_i, to every S_j at the same time receiving TVs produced by others.
2. Upon the reception of some TV_j, S_i updates TM_i following the same procedure described in section 9.2.1.
3. If a site S_i does not receive some TV_j, S_i will have an inaccurate *status* estimation of site S_j and could announce, by starting a log sender election phase, its willingness to propagate some LFUs.
4. However sites register themselves to multicast sessions on the basis of their own TV. As a consequence, propagation will take place if and only if some site will be interested in the propagated LFUs, thus saving bandwidth resources.

[1] max and min among the set of connected sites

Our approach offers some advantages also thanks to the adoption of reliable multicast technology [HAA99]. In fact sites receive TVs transmitted from their respective owners (*status* transmission is not epidemic). As a result, there are three immediate consequences:

1. Bandwidth savings with respect to the unicast solution in which sites exchange the whole matrix (reduced amount of transmitted *status* information);

2. In absence of *status* transmission failures, each site will correctly estimate the reception *status* of the whole system, whereas in the unicast epidemic approach each site maintains just a conservative estimate;

3. Even in presence of *status* transmission failures, the adopted reliable multicast protocol avoids useless transmission by requiring explicit session registration for the *LFU* reception (reduced amount of transmitted *log update* information).

9.2.4 Log Sender Election Algorithm

In order to limit concurrent multicast retransmissions of the same LFUs by multiple sites due to the epidemic model, a distributed log sender election algorithm is adopted. For the sake of simplicity we will assume in our description of the algorithm that just one LFU at a time will be propagated. More complex cases can be easily reduced to this one.

During the log sender election phase, sites are all listening to multicast announcements for LFUs propagation. Each time a site S_i wants to propagate an LFU l, it will multicast to all sites S_k $k = 1, \ldots, n$ an announcement message, identifying:

- The site which generated l;
- The timestamp of l;
- The starting time for the transmission (randomly determined by a backoff timer).

Upon the reception of such an announcement, any site S_k interested in receiving the LFU l will register in the multicast session. Otherwise if any site S_j was planning to transmit the same LFU l as S_i and its time to transmission[2] is larger than the one deducible from the received announcement, S_j simply gives up its own propagation phase. Multicast transmissions of different LFUs are not in competition.

It is worth noticing that, as long as log updates are sent in distinct multicast sessions, our replica management algorithm completely solves the duplicate updates problem that affects the unicast solution.

[2] given by current time + backoff timer

9.2.5 Reference Scenario

We introduce now the reference Data Grid environment we adopted (see Fig8.4).

The reference Grid application we refer to (e.g. genomic research) produces on average 1 GB new data per day per site, which can be stored using standard database technology. Once locally committed, stored data will usually remain unchanged (i.e. it is mostly read-only). At regular intervals (e.g. daily at 24:00h) each site propagates its new data to every other site. Such information is sent as a number of update files per site containing database operation logs. Received log files can be replayed and stored for later use.

9.3 Prototype Implementation

Gedec is the Java prototype of our replica update propagation system. It relies on the services offered by JMFTP [DGLS02a] for reliable file transfer.

The prototype software architecture contains the core modules interfacing to the multicast network layer.

Software modules can be logically divided into two categories: data sending and data receiving modules. In a multi-master system each site must execute modules of both categories. Gedec is composed of the following main modules (see fig.9.2):

- ***Timertask***: schedules the TV and log update operations synchronously with all other sites;
- ***TV(TM)Handler***: manages the read/write access to each Timestamp Vector (Matrix);
- ***Sender***: manages data update pools and selects data to be sent via the JMFTP-Server module;
- ***Listener (Receiver)***: manages transmission announcements reception and selects which data the JMFTP-Client modules will receive;
- ***JMFTP-Server***: reliably multicasts announces and data to all members of the multicast tree;
- ***JMFTP-Client***: receives announces and data from any JMFTP-Sender.

Each TV fields is 2 bytes long in order to fit as much of the TV as possible in a single UDP packet payload. This choice reduces the time range to 2^{16} time intervals, but in addition each packet carries the base time for all these interval counters in standard format (i.e. DD:MM:YYYY:HH:MM:SS).

Each module is implemented as an independent Java Thread following the Java Thread Design Guidelines [Lea01] for deadlock avoidance.

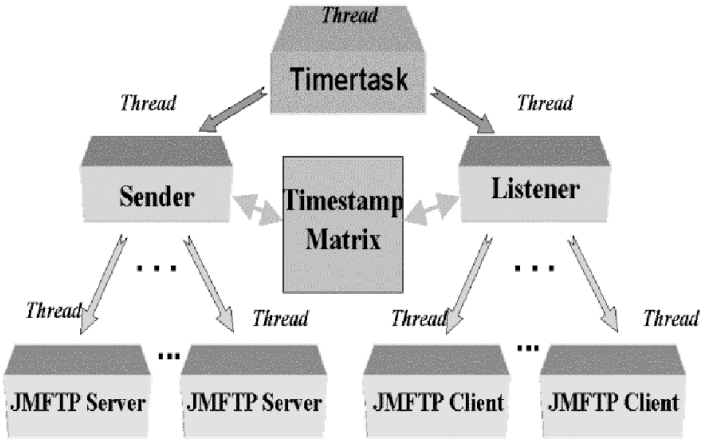

Figure 9.2: **Gedec software architecture**

9.4 Gedec Evaluation

9.4.1 Testing Activity

The testing activity we describe includes the performance comparison of two different lazy multi-master replica update distribution strategies adopting GridFTP [VSF02] and Gedec. The objective is to show the efficiency of the replica update propagation strategy we adopted with respect to an actual solution.

We evaluate our multicast-enabled propagation system and compare it with a smart distribution strategy using actual Grid technology. We label the latter as GridFTP since it is strictly tied to the performance of such protocol.

Performance is evaluated with respect to a varying number of replica sites. The Total Distribution Time is measured as the time spent by the overall system to distribute all LFUs to all sites. The bandwidth consumption for the replica update is measured as the average data volume sent by each site during the daily update phase.

The testbed consists of n sites connected using full-duplex 100Mb/s links and Dummynet [Riz97]. We adopted Dummynet to introduce WAN (emulated) links in our scenario, in order to evaluate the behavior of the propagation strategies on a mixed network type.

The following general assumptions hold (see reference application scenario (9.2.5) above):

- average update data creation rate 5.7 Mb/min per site;
- induced average packet loss rate 0.5%;

In other words 5.7Mb/min, i.e. 1GB of Log File Update data per day are created by each site, that have to be propagated daily to every other site. A Log File Update is itself a file that is transferred in order for the sites receiving it to update their replicas.

GridFTP performance is estimated by using 8 parallel ftp streams per transfer. In order to shorten total distribution time, a single LFU file of size 1GB is sent using a multi-step strategy where (supposing sites are numbered from 1 to n): i) in step 1 every site m transfers data to site $(m+1)$ *mod* n; ii) in step 2 every site m transfers data to site $(m+2)$ *mod* n; iii) in step j every site m transfers data to site $(m+j)$ *mod* n; iv) up to step $n-1$ included.

Such a strategy is adopted instead of performing simultaneous all-to-all ftp transmissions because in the latter case $2n$ data flows would have to be handled (n incoming, n outgoing) by each site, and that would not scale both for network and for site capacity.

Gedec behavior is as described above (see section 9.2.3) and characterized by the following:

- TVs multicast at fixed time intervals of 5 minutes;
- 102 LFUs of size 10MB multicast by each site;
- LFU multicast rate 2.0Mb/s (that is 0.25MB/s);

It is clear we are comparing a slow constant rate reliable multicast distribution strategy with a faster reliable unicast (ftp) approach. According to the vast literature ([VSF02], [VS02],[EKK+02], [VSF02], [KLSS04], [KKL04]) the parallel ftp rate is expected to be comprised between 0.5 and 10 MB/s, whereas the multicast emission rate is set to 2.0Mb/s. In fact, in our scenario multiple parallel ftp reaches a 1-3MB/s rate.

Figure 9.3 shows the average time the replication strategy spends in propagating LFUs among all replica sites. Our results show that, on the one hand, Gedec is slower than GridFTP in updating systems when n is small. This is due to the fixed data emission rate of JMFTP. On the other hand, Gedec performance is comparable to GridFTP and better as the number of sites n increases.

As regards bandwidth consumption, figure 9.4 shows the difference between Gedec (multicast) and GridFTP (unicast) in this area. The available network bandwidth can be a bottleneck for data transfers. In our scenario this can start impacting on constant rate multicast performance when n is approximately greater than 50.

Furthermore Gedec, by multicasting *status information* and by announcing transmissions, avoids unnecessary log file update transfers that may happen using a plain multicast epidemic distribution strategy.

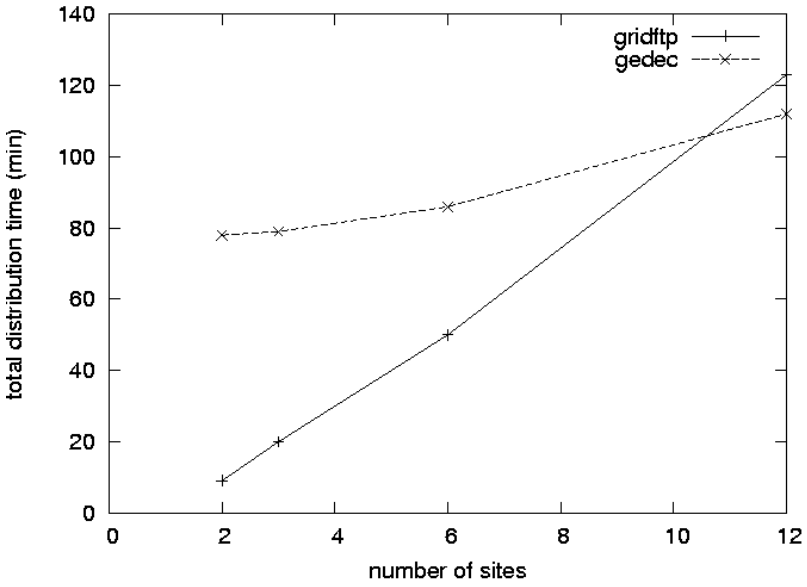

Figure 9.3: **Updates propagation time**

Compared to unicast-based approaches, Gedec can reduce the amount of transmitted *log update* information and decrease total distribution time, thus potentially increasing system scalability with respect to the number of sites.

A consideration on the improvement of Gedec over GridFTP for file distribution is that the impact of packet loss is much stronger on ftp performance (due to the TCP congestion control behavior) than on JMFTP's. This is so since ftp is known to suffer [LXC03] from high packet loss rates.

Thus the performance gain of Gedec over GridFTP is stronger as the packet loss rate increases. Such a feature makes it particularly suitable for high packet loss scenarios.

9.4.2 Conclusion

The main result of this section is the elaboration and evaluation of a new system for update propagation in the context of multi-master replica management systems.

Some of the main peculiarities of Gedec with respect to previous solutions (e.g. [HAA99], [AAS97]) are:

- Multicast allows for fast direct distribution of *status* information, represented by Timestamp Vectors, among sites.
- Site *status* estimation is more precise than using the unicast epidemic approach, where each site has a conservative estimate of the global *status*.

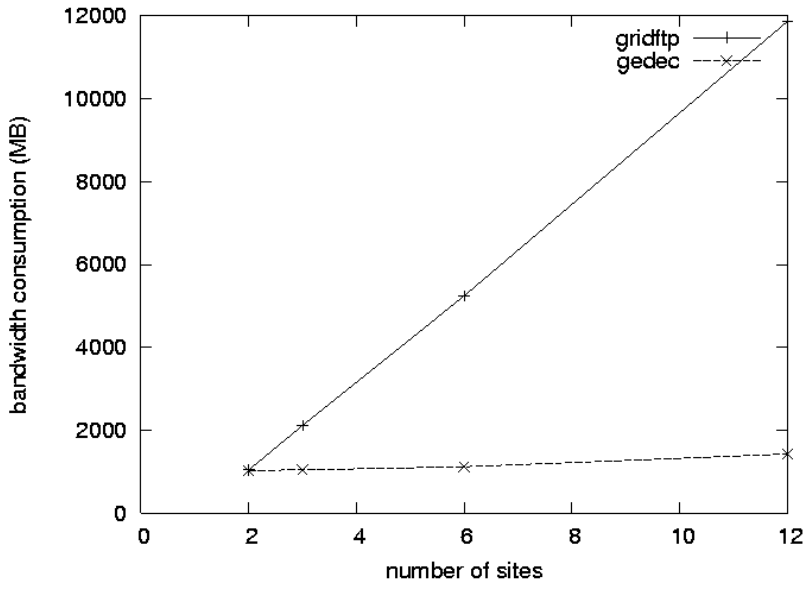

Figure 9.4: **Network utilization**

- Propagation (Total Distribution) time is reduced in many scenarios thanks to JMFTP reliable multicast.

- Bandwidth consumption is lower than solutions that adopt unicast.

Furthermore, since each log file update is sent in a distinct multicast session, sites need to register explicitly in order for the sender to start transmitting. Thus, the approach described above solves the duplicate update problem that affects epidemic propagation models.

We can conclude that the adoption of multicast technology in the replica management problem is feasible and can reduce the traffic generated by *log update* transfers among replicas. The proposed system proves effective in our testbed while further broader evaluation is planned.

As a final consideration, it would be interesting to study the integration of some of the experience we gathered with a state of the art complex replica management system such as Reptor.

9.5 Web Service UDDI Issues

This is a challenging area where multicast technology can play a positive role. Furthermore, Grid technology has shifted towards the adoption of Web Service tech-

nology for service integration (see OGSA and GT 4). In this section we describe a solution for smart selection of web services replicas[BGGL04] we proposed.

This sections define the main logical functionalities and the software architecture of a framework that overcomes the limits of UDDI in fully exploiting the advantages offered by dynamic binding. The framework, relying on the services offered by remote monitoring agents, makes it possible for UDDI Registries to associate a value of convenience to different implementations of the same tModel. This value will lead Service Requestors in the choice of implementations to be contacted and invoked.

UDDI Registries with this enhanced behavior still expose an interface compliant to the standard UDDI specification, thus being fully and transparently integrable in the UDDI infrastructure. The framework fills the gap existing between approaches to the discovery of Web Services that do not deal with any aspect of their performance and more complex solutions that guarantee a certain level of quality in providing services.

The definition of a general *Web Service Architecture* [Kre01] is based upon the interaction of three main roles, as shown by figure 9.5:

- *Service Providers*, responsible for compiling *service descriptions*, i.e. all the informations required to interact with a Web Service, *publishing* them to one or more Service Registries and dealing with Web Services invocation messages from Web Service Requestors;
- *Service Requestors*, responsible for *finding* service descriptions into Service Registries and *binding* to Web Services;
- *Service Registries*, i.e. searching registries whose main responsibility is to advertise published Web Service Descriptions to be found by Web Service Requestors.

Going into deeper details a service description mainly consists of two separate parts:

- An *abstract interface*, that logically describes the Web Service in terms of the messages that have to be exchanged between a Service Requestor and a Service Provider;
- An *implementation interface*, containing all the implementation details needed to contact and invoke the Web Service, such as one or more points of access, usually URLs, to different implementations of the abstract interface. These implementations will be referred to as *Web Service Instances* in what follows.

The main advantage of this separation is abstract interface reusability: points of access in several implementation interfaces can share the same abstract interface.

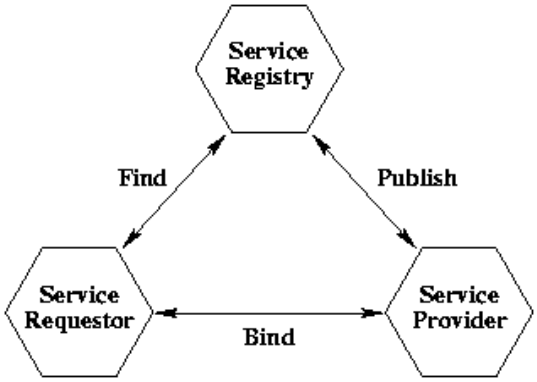

Figure 9.5: **Web Service architecture**

At present, the *publish* and *find* operation can be performed using a range of technological solution, as DISCO/ADS, WSDL Repository, ebXML, UDDI and others. Among them the UDDI (*Universal Description, Discovery, and Integration*) specification [UDD03] differs from other proposals by virtue of the substantial commitment from industry partners to adopt and implement this technology in their core business. As a demonstration of this commitment, companies such as SAP, IBM and Microsoft, have launched jointly operated UDDI Registries on the Web [UDD01].

According to the aim of our research work, a relevant UDDI functionality is the following: an *UDDI Registry* can be queried for the list of point of access implementing a given abstract interface, *tModel* in the UDDI terminology. The selection of a particular point of access is delegated to Service Requestors and can be accomplished either at development time, *static binding*, or at execution time, *dynamic binding*.

In order to fully exploit the advantages offered by *dynamic binding*, the UDDI Technical Committee suggests the following invocation pattern [UDD03]: the Web Service Requestor should cache the list of points of access for the tModel of interest. If all the points of access are unsuccessfully tried, the Web Service Requestor should query again the UDDI Registry for a fresh list. The limit of this paradigm is intrinsic to UDDI Registries standard functionalities. In fact they do not provide any information on the *convenience* of selecting one access point instead of another. In this section we report on the research activity conducted by Netlab, the network and multimedia laboratory of IASI, with the aim of developing a framework, called *Enhanced UDDI* in what follows, that overcomes this limit.

The framework fills the gap existing between approaches to the discovery of Web Services not dealing with any aspect of their performance, such as that implemented by UDDI, and more complex solutions introduced to guarantee a certain level of quality in providing services. In fact *Enhanced UDDI* avoids the main drawbacks of the former, such as the selection of Web Service Instances that are overwhelmed by

service requests or that are not available. At the same time it reduces the difficulties intrinsic to the latter, such as deriving what it takes to provide the QoS which is offered or that has been agreed upon, or managing a service at different QoS levels on the same infrastructure.

To overcome the limits of UDDI and to effectively support the invocation pattern suggested by the UDDI Technical Committee, *Enhanced UDDI* makes it possible for UDDI Registries to associate a value of convenience to access points. This value is computed on the basis of the information collected by remote agents monitoring some aspects of the performance achieved by Web Service Instances. Points of access then appear in the list returned from UDDI Registries to Service Requestors sorted from the most to the least convenient.

Enhanced UDDI registries still expose an interface compliant to the standard UDDI specification thus being fully and transparently integrable in the UDDI infrastructure.

The adoption of this framework is suggested whenever service requestors do not intend to explicitly deal with QoS aspects or service providers have no convenience in building up the software and hardware infrastructures needed to manage the same aspects, but are nevertheless interested in avoiding providing services of bad quality to their customers.

This section also describes the main design and implementation issues concerning a first Java prototype of the *Enhanced UDDI*. The prototype has been tested from a functional correctness point of view by implementing:

- A light agent monitoring availability, time of completion and probability of success of Web Service Instances;
- An Enhanced UDDI Registry, polling the remote agents and associating to each Web Service Instance a value of convenience computed as a weighted average of monitored parameters.

9.5.1 Main Functional Aspects of Enhanced UDDI

The main functional aspects of *Enhanced UDDI* could be introduced referring to the following model of a real scenario (see figure 9.6): *n* Web Service Instances, all implementing the same tModel, are available on *n* different Web Servers (Service Providers). The *n* Web Service Instances are all registered to the same UDDI Registry.

As already stated, UDDI does not provide any information on the *convenience* of selecting one access point instead of another. In order to overcome this limit, *Enhanced UDDI* relies on the services offered by a remote monitoring agent, referred to as *Enhanced UDDI Monitoring Agent* in what follows, that runs on each Web Server. As shown in figure 9.7, the Enhanced UDDI Registry polls the monitoring agents in order to receive the information they collected on the instances behaviors and to

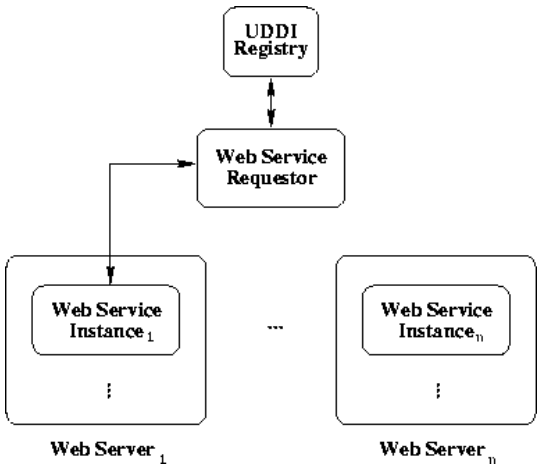

Figure 9.6: **A model for a real scenario**

associate a value of convenience to each access point. The Enhanced UDDI Registry computes these values to sort access points from the most to the least convenient. Finally, the ordered list is ready to be returned to Web Service Requestors.

At present, from Web Service Requestors' point of view, *Enhanced UDDI* still exposes the same interface of a regular UDDI registry, but it will be soon extended to able them to specify the sorting rule to apply. The following two sections describe the software architecture of the two main components of the framework, the *Enhanced UDDI Monitoring Agent* and the *Enhanced UDDI Registry*.

9.5.2 Enhanced UDDI Monitoring Agent

The task of the Enhanced UDDI Monitoring Agent is to monitor the behavior of Web Service Instances with respect to a given set of m measurable parameters, such as service availability, response time, probability of success, etc.

As shown by figure 9.8, the main components of an Enhanced UDDI Monitoring Agent are the *Logs Generator*, the *Statistics Generator* and the *Statistics Provider*.

The *Logs Generator* is the integration component between the Web Server and the Agent. It filters the traffic addressed to and generated from the Web Server and processes service requests and responses in order to measure the values of measured parameters. In addition, the Logs Generator interacts, whenever possible, with the monitoring components provided by the Web Server. The generated information is locally maintained into the *Logs Archive*.

At fixed time intervals the *Statistics Generator* derives, from the *Logs Archive*, the average values on such intervals for all parameters. These average values are then arranged in a vector and stored into the *Statistics Archive*. Finally, the *Statistics*

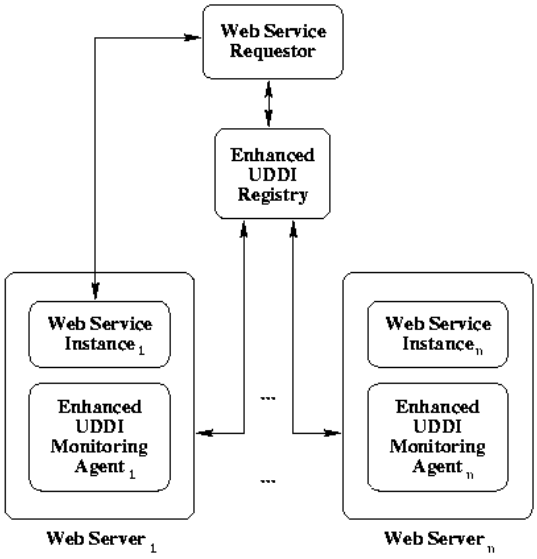

Figure 9.7: **Main components of the framework**

Provider makes the vectors available to the Enhanced UDDI Registry.

9.5.3 Enhanced UDDI Registry

The Enhanced UDDI Registry extends the functionalities of currently available UDDI Registries with respect to all those operations related to the selection of access points and the publishing of Web Services. At the same time it exposes an interface compliant to the standard UDDI API, thus being fully and transparently integrable in a UDDI infrastructure.

Figure 9.9 shows a first level description of the Enhanced UDDI Registry software architecture. The UDDI Extender accomplishes two main tasks:

- It is responsible for implementing the extensions provided by the Enhanced UDDI Registry;
- It is the registry interface towards Service Requestors.

Whenever needed, the UDDI Extender delegates the execution of UDDI operations to an internal UDDI Registry. The *Statistics Archive* maintains a temporal sliding window on the content of the corresponding archive inside all Enhanced UDDI Monitoring Agents. The Archive is updated by polling the Enhanced UDDI Monitoring Agents. The UDDI Extender relies on this information to associate values of convenience to access points. To accomplish its task the Enhanced UDDI Registry

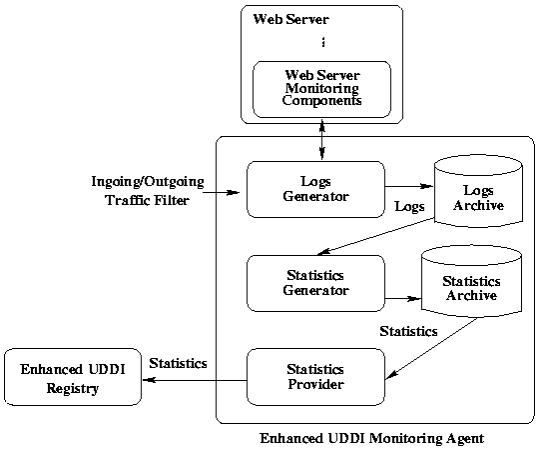

Figure 9.8: **The Enhanced UDDI Monitoring Agent software architecture**

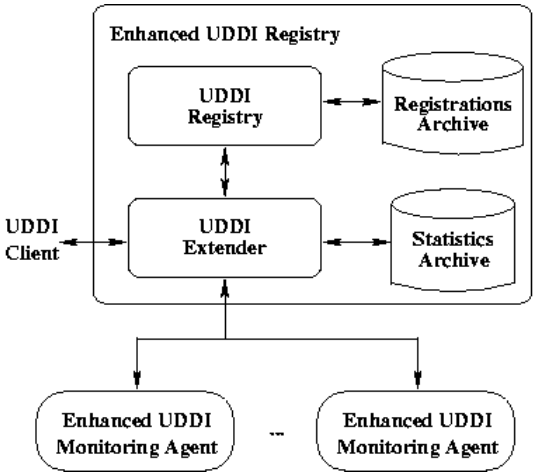

Figure 9.9: **The Enhanced UDDI Registry software architecture**

also maintains a list indicating which of all Web Service Instances are being monitored. In this way it is possible to register with the Enhanced UDDI Registry also those Web Service Instances which are not being monitored.

As shown in figure 9.10 four main software modules define the internal architecture of the UDDI Extender:

- *Request Catcher*. It intercepts the UDDI client requests. For non-extended operations the Request Catcher simply plays the role of a proxy between the UDDI Client and the UDDI Registry. Otherwise, it delegates to the UDDI Extension Manager the accomplishment of the operation.

- *Agents Manager*. Its main task is to retrieve and update the information concerning the behavior of the monitored Web Service Instances. It periodically polls the Enhanced UDDI Monitoring Agents, downloading all newly collected vectors and storing them into the Statistics Archive.

- *Point of Access Evaluator*. Its main task is to associate a value of convenience to each registered point of access. More in details, given an access point, it retrieves into the Statistics Archive the collection of all vectors related to that access point. It then applies to such collection a given evaluation function $f : V^n \rightarrow \mathbf{R}$, where V is the space of m-dimensional vectors, defined as follows:

$$f(\mathbf{v_1}, \mathbf{v_2}, \ldots, \mathbf{v_n}) = \sum_{k=1}^{n} w_k \left(\sum_{i=1}^{m} c_i x_{i,k} \right)$$

with

$$\sum_{k=1}^{n} w_k = \sum_{i=1}^{m} c_i = 1, w_k, c_i, x_{i,k} \geq 0$$

where

$$\mathbf{v_k} = \begin{pmatrix} x_{1,k} & x_{2,k} & \ldots & x_{m,k} \end{pmatrix}, k = 1, 2, \ldots, n$$

Larger values of the weights w_k are assigned to more recently collected vectors, while larger values of weights c_i are assigned to more *significant* vector components.

- *UDDI Extension Manager*. Given a tModel, it queries the UDDI Registry for the points of access to the Web Service Instances implementing that tModel. The UDDI Extension Manager delegates to the Point of Access Evaluator the evaluation of a value of convenience for each of these access points.

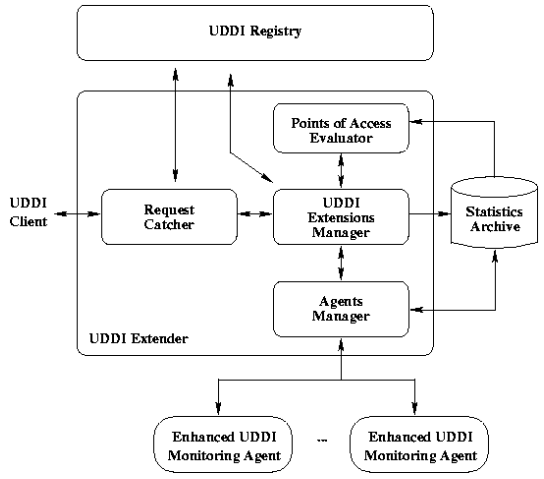

Figure 9.10: **The UDDI Extender software architecture**

9.5.4 Web Service Instance registration issues

Since the Enhanced UDDI Registry exposes an interface compliant to the standard UDDI specification, the registration modalities are not affected from the UDDI client point of view.

Also from an internal perspective, the registration phase is accomplished by Enhanced UDDI Registries similarly to UDDI Registries. The main difference is that, for each newly registered Web Service Instance, the Enhanced UDDI Registry tries to contact the Enhanced UDDI Monitoring Agent, if any, for that instance: if the Agent is found, that instance is added to the list of monitored Web Service Instances.

The last problem to deal with is how to locate the Enhanced UDDI Monitoring Agent. The proposed solution makes it possible to trivially accomplish this task. In fact if

$http://domainName/localPath/WSIName$

is the URL of a monitored Web Services Instance, the agent will have to be available at:

$http://domainName/eUDDIMAgent$.

Thus, the URL of the Enhanced UDDI Monitoring Agent can be easily derived from the URL of any of the Web Services Instances it monitors. As a consequence, it is not necessary to explicitly notify Enhanced UDDI Registries about the URLs of the Enhanced UDDI Monitoring Agents they will rely on.

Moreover, this mechanism also supports registrations induced by the UDDI Registry replication process. In fact, the Enhanced UDDI Registry deals with replicated registrations by checking whether they refer to monitored Web Service Instances or not.

9.5.5 Prototyping Activity

In order to test the feasibility of the proposed solutions, a Java prototype of the *Enhanced UDDI* is being developed. In what follows we briefly describe the main design and implementation issues that characterize the prototype.

9.5.6 Enhanced UDDI Monitoring Agent

At present, a first preliminary version of Enhanced UDDI Monitoring Agent has been implemented. For every Web Service Instance running on the same Web Server hosting agent, the last one is able to measure the following QoS parameters:

- *Availability* - It is the probability the Web Service Instance is up. It is measured as $A = upTime/(upTime + downTime)$, where $upTime$ and $downTime$ are the total time the system has been up and down during the measurement period, respectively.

- *Time of Completion* - It is an evaluation of the speed in completing a service request. It is measured as $ToC = (leavingTime - deliveringTime)$, with $deliveringTime$ the time at which a SOAP service request is delivered to the SOAP engine, and $leavingTime$ the time at which the related SOAP service response leaves the SOAP engine.

- *Probability of success* - It is defined as $(1 - ProbabilityOfFailure)$. $ProbabilityOfFailure$ is measured as $PoF = SoapFaults/SoapResponses$, where $SoapFaults$ is the number of Web Service Instance's responses containing the *Faults* XML element in the SOAP envelope and $SoapResponses$ is the total number of SOAP responses.

The Enhanced UDDI Monitoring Agent has been implemented as a module integrated with the Apache Axis engine [Foub]. Axis is an implementation of the SOAP submission to W3C whose extensible filtering functionality (Axis handlers) makes it possible to easily extract information from SOAP request and response messages.

Finally, the information collected by the Enhanced UDDI Monitoring Agent is made available to Extended UDDI Registries through a Web Service.

9.5.7 Enhanced UDDI Registry

At present this component is being developed as an extension of jUDDI [Fouc], a Java implementation of the UDDI specification for Web Services. jUDDI has clearly defined and extendable classes and has been adopted by the Apache Software Foundation [Foua], thus guaranteeing further open development. For these reasons jUDDI is particularly well suited for the Enhanced UDDI Registry implementation.

Referring to Figure 9.10, the Request Catcher will intercept UDDI requests and reroutes to the UDDI Extension Manager UDDI requests such as *find_binding*, *find_tModel*, *get_bindingDetail*, *get_serviceDetail*, *get_tModelDetail*, *delete_tModel*, *save_binding*, *save_service* and *save_tModel*.

In order to perform the first functionality tests, evaluation functions like the following will be used:

$$f(\mathbf{v_1}, \mathbf{v_2}, \mathbf{v_3}, \mathbf{v_4}) = \begin{pmatrix} 1/8 & 1/8 & 1/4 & 1/2 \end{pmatrix} M$$

where

$$M = \left(\begin{pmatrix} A_1 & 1/ToC_1 & T_1 \\ A_2 & 1/ToC_2 & T_2 \\ A_3 & 1/ToC_3 & T_3 \\ A_4 & 1/ToC_4 & T_4 \end{pmatrix} \begin{pmatrix} 1/3 \\ 1/3 \\ 1/3 \end{pmatrix} \right)$$

.

Referring to the general formula presented in paragraph 4, $n = 4$ denotes that f considers only the last four retrieved vectors. Among those the most relevant is the freshest (weight *1/2*), while, looking inside each vector, the three QoS parameters are equally considered (weight *1/3*). Finally, it is worth noting that all numerical values stored into the vectors are normalized as real number with values in $[0,1]$ interval.

The results of a large number of tests will be collected in order to define the most effective evaluation functions.

The Enhanced UDDI relies on the services offered by remote agents monitoring some aspects of the performance of Web Service Instances. On the basis of the collected information, UDDI registries compute and associate a value of convenience to access points.

At present, the points of access are sorted from the most to the least convenient and returned to the *Service Requestor*. UDDI registries with this enhanced behavior still expose an interface compliant to the standard UDDI specification, thus being fully and transparently integrable in the UDDI infrastructure.

The Enhanced UDDI is useful whenever service requestors do not intend to explicitly deal with QoS aspects or service providers have no convenience in building up the software and hardware infrastructures needed to manage the same aspects, but are nevertheless interested in avoiding providing services of bad quality to their customers. We also described the main design and implementation issues concerning a first Java prototype of the framework.

Current research and experimentation activities are focused on the:

- Development and testing of Enhanced UDDI Registry component;
- Experimental analysis of the benefits achievable by Enhanced UDDI Registries from the Service Requestors point of view;
- Definition of a set of effective monitored parameters;

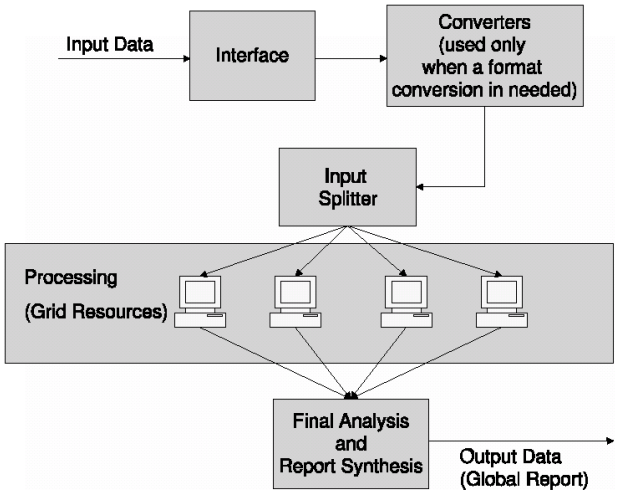

Figure 9.11: **Jet-Lag architecture**

- Experimental validation of effective policies and protocols according to which Enhanced UDDI Registries and Enhanced UDDI Monitoring agents would exchange information;

Future research will study the internal Enhanced UDDI behavior with respect to a complex scenario where the Web Services space is partitioned on several instances of Enhanced UDDI Registries.

9.6 Jet-Lag

In this section we briefly describe an investigation aimed at gathering experience and in order to test a real-world EU Data Grid [Dat] application.

Jet-Lag [LP05] is a Grid application performing analysis of log files that conform to several different standard formats. These files can be produced by firewalls, IDSs, mail and web servers. In order to exploit the potential execution parallelism of the Grid, a software architecture has been designed and developed which splits log files into smaller fragments which are then analyzed independently, possibly in parallel, using computing resources available in a Grid computing environment (see fig. 9.11). Jet-Lag functionality and performance tests have been performed on a DataGrid testbed. First experimental results show that Grid log analysis with the proposed software architecture is feasible and effective. Performance, as expected for a first Java prototype implementation, is not yet optimal (see fig. 9.12), but we are working to solve specific issues and on the generalization of the original approach.

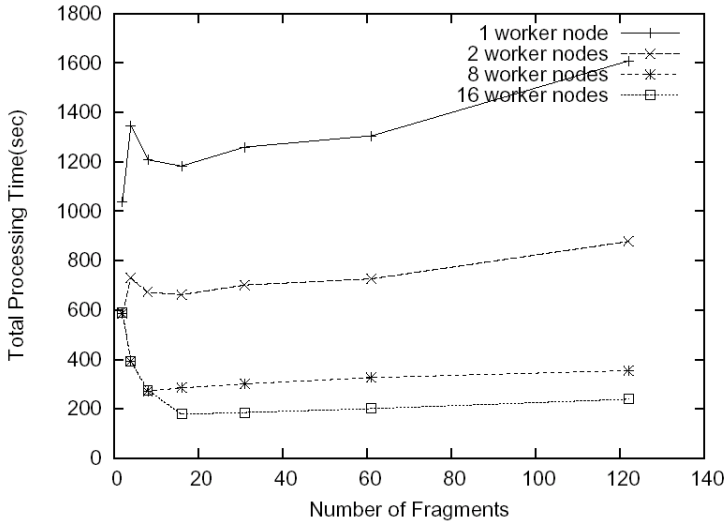

Figure 9.12: **Jet-Lag performance**

Chapter 10

Conclusion and Further Work

Multicast transmission allows bandwidth-efficient data exchange among members of a group. IP multicast has proven to be a technology more scalable than unicast, but nevertheless limited, concerning reliability, security and scalability aspects in the real world. Solutions proposed in literature are not complete and have not had a wide acceptance by the research and the engineering communities.

In our aim to study, improve and validate reliable multicast protocol solutions, we have created a Java software framework containing building blocks that has been extended and combined to implement the desired protocol features and to collect experimental data.

Java has been adopted as a reference platform by our research group. It guarantees extreme portability and is particularly suitable for highly scalable multicast scenarios. Furthermore, along the Ph.D. years, the choice of Java has proven increasingly consistent with the market evolution. In fact, nowadays we have millions of Java-capable devices available everywhere (PCs, cell-phones, PDAs, set-top-boxes).

Our framework comprises JMFTP, a module providing high-performance scalable reliable multicast data transfer. We analyzed the throughput and robustness of JMFTP and individuated trade-offs among protocol parameters.

JMFTP proves extremely robust with respect to data loss and delay, rendering it particularly suitable for satellite networking.

Another module of our framework is the implementation of a secure key exchange protocol named MTLS, providing secure group membership management and session key encryption of data.

We integrated cryptography and key exchange with JMFTP obtaining a secure reliable multicast transport protocol transferring bulk data from one sender to multiple receivers throughout the Internet. Performance test were performed that allowed us to evaluate the impact of security on performance.

We then adopted the above framework to support data replica synchronization in a Grid computing infrastructure. Test results on Gedec, our replica update distribution

system, have shown that such a Grid application may benefit from the adoption of the proposed replication model while further research is being done on the subject.

With Gedec, we gathered experimental evidence of how high-performance tasks such as the scalable replica synchronization can be done using Java. We also showed that complex real-world tasks such as distributed Grid computing applications benefit from the adoption of our framework and from a reliable multicast distribution model.

10.1 Contribution and Future Work

In this thesis we collected research investigations on the main topics in multicast transmission: performance, security and applications. Throughout our work we have given evidence of the feasibility of a high-performance Java secure reliable multicast protocol; we have gathered experimental data on real protocol implementations that can be used as a reference for further work.

The tests we performed lead us to discover and measure trade-offs among security, performance and reliability. The software framework we created contains building blocks allowing for multicast protocol implementation, monitoring and performance measurement;

We have also proposed the application of our framework to the Grid replica synchronization problem and have studied the performance issues of such a solution.

Following the current research trends we have investigated Web Services issues and proposed a solution to the Web Service instance selection problem. Web Service technology is even more interesting since it has been adopted as the next Grid paradigm for its flexibility. The OGSA Framework and the next Globus Toolkit middleware are taking advantage of the application of the Web Service paradigm to Grid architectures. We are currently working in the same direction and plan to keep contributing to this research area. Other main research directions we are following include:

- Investigating the performance of the new multicast protocol features whose support has been added to our framework, such as multichannel, multirate and flexible real-time transmission support.
- Experimenting with real Active Reliable Multicast Routing Protocol that has been implemented via an extension to the Linux packet filtering functionality.
- Enhancing the key redistribution algorithm used for MTLS and analyzing tests results on wider more complex scenarios.
- Investigating the extension of the Web Service paradigm to other solutions we proposed.
- Integrating Gedec with existing replica management solutions and perform tests on a larger scale.

- Applying multicast to the UDDI replication problem.
- Studying wireless multicast issues.
- Integrating components with the Globus Toolkit version 4.

10.2 Published Papers

- A Java implementation of a reliable multicast file transfer protocol:design and evaluation. In Proceedings of *Internet and Multimedia Systems and Applications* [BDGL01].
- A Java architecture for secure reliable multicast data transfer: performance evaluation. In Proceedings of *Communication Internet and Information Technology* [DGLS02a].
- Performance evaluation of a reliable and secure multicast bulk data transfer application in Java. In Proceedings of *Computer Measurement Group Workshop* [DGLS02b].
- Smart dynamic selection of web service access points. In Proceedings of *Parallel and Distributed Computing and Systems* [BGGL04]
- A reliable multicast approach to replica management for Grids. In Proceedings of *Parallel and Distributed Computing and Networks* [GL05].
- Jet-Lag Java heterogeneous log analysis on the Grid: architecture, implementation and performance evaluation. In Proceedings of *Parallel and Distributed Computing and Networks* [LP05].

10.3 Software Implementations

- JMFTP framework and protocol.
- MTLS component.
- NJMTP protocol.
- GEDEC system.
- Jet-Lag application.
- eUDDI system.

Appendix A

Acronyms

ACK : ACKnowledgment
AGCS: Alternate Group Communication Service
ALM : Application Level Multicast
API : Application Program Interface
ARM : Active Reliable Multicast
ARP : Address Resolution Protocol
AS : Autonomous Systems
BGP : Border Gateway Protocol
BGMP: Border Gateway Multicast Protocol
BKR : Batched Key Retransmission
BPM : Blind Pushing Multicast
CBT : Core Based Tree
CE : Computing Element
DCM : Distributed Core Multicast
DCR : Distributed Core Router
DFS : Distributed File System
DISEC: Distributed Security
DPSS: Distributed Parallel Storage System
DTU : Data Transmission Unit
DVMRP: Distance Vector Multicast Routing Protocol
DyRAM: Dynamic Replier Active Reliable Multicast
GARA: Grid Advanced Resource reservation and Allocation
GASS: Grid Access to Secondary Storage
GC : Group Key Controller
GDMP: Grid DataMirroring Package
GC : Group controller
GM : Group Member
GO : Group Owner

GRAM: Globus Resource Allocation Manager
GSA : Group Security Association
GSI : Grid Security Infrastructure
GT : Globus Toolkit
HEP : High Energy Physics
HPSS: High Performance Storage System
HTT : Host Transfer Time
IDS : Intrusion Detection System
IETF: Internet Engineering Task Force
JRMS: Java Reliable Multicast Services
KEK : Key Encryption Key
KS : Key Server
LAN : Local Area Network
LFU : Log File Unit
LKH : Logical Tree Hierarchy
LRMP: Lightweight Reliable Multicast Protocol
MAC : Message Authentication Code
Mb : Megabit
MB : Megabyte
MBONE: Multicast BackbONE
MCM : Multiple Channel Multicast
MDS : Meta Directory Service
MFTP: Multicast File Transfer Protocol
MHP : Multimedia Home Platform
MIP : Multicast Internet Protocol
MOSPF: Multicast Open Shortest Path First
MTLS: Multicast Transport Layer Security
NACK: Negative ACKnowledgement
NAK : Negative AcKnowledgement
NAPP: Negative Acknowledge with Periodic Polling
NETBLT: NEtwork BLock Transfer
NORM: Nack Oriented Reliable Multicast
OGSA: Open Grid Services Architecture
OGSI: Open Grid Services Infrastructure
OO : Object Oriented
PBR : Periodic batched group rekeying
PIM : Protocol Independent Multicast
QosMIC: Quality of Service-sensitive Multicast routIng protoCol
RB : Resource Broker
RC : Replica Catalogue
RINA: Receiver Initiated Negative Acknowledge
RM : Replica Manager

RMS : Replica Management System
RMP : Reliable Multicast Protocol
RMT : Reliable Multicast Transport
RMTP: Reliable Multicast Transport Protocol
RP : Rendevouz Point
SAKM: Scalable Adaptive Key Management protocol
SE : Storage Element
SLIM: Self-configuring Lightweight Internet Multicast
SRM : Scalable Reliable Multicast
STORM: STructure Oriented Resilient Multicast
TLS : Transport Layer Security
TM : Timestamp MAtrix
TMP : Topology Management Protocol
TMTP: Tree-based Multicast Transport Protocol
TRAM: Tree-based Reliable Multicast Protocol
TRT : Total Re-keying Time
TTP : Trusted Third Party
TTT : Total Transfer Time
TV : Timestamp Vector
UDDI: Universal Description Discovery and Integration
WAN : Wide Area Network
WKA-BKR: Weighed Key Assignment Batch Key Retransmission
WN : Worker Node
WSRF: Web Services Resource Framework

Bibliography

[AAS97] D. Agrawal, A. E. Abbadi, and R. C. Steinke. Epidemic algorithms in replicated databases (extended abstract). In *Proceedings of the sixteenth ACM SIGACT-SIGMOD-SIGART symposium on Principles of database systems*, pages 161–172. ACM Press, 1997.

[AB04] B. Adamson and C. Borman. Nack-oriented reliable multicast protocol (norm) - internet draft http://norm.pf.itd.nrl.navy.mil/draft-ietf-rmt-pi-norm-09.pdf, 2004.

[All] Globus Alliance. Globus - http://www.globus.org.

[All01] B. Allcock. Gridftp protocol. internet draft - http://www.globus.org /datagrid/deliverables/C2WPdraft3.pdf, 2001.

[ANS03] A. Adams, J. Nicholas, and W. Siadak. Protocol independent multicast - dense mode (pim-dm): Protocol specification (revised). internet draft - http://netweb.usc.edu/pim/internet-drafts/draft-ietf-pim-dm-new-v2-04.txt, 2003.

[Bal97] T. Ballardie. Core based trees (cbt version 2) multicast routing. RFC 2189, September 1997.

[BB99] L. Blazevic and J. Le Boudec. Distributed core multicast (dcm): a multicast routing protocol for many groups with few receivers. *ACM SIGCOMM Computer Communication Review*, 29(5):6–21, 1999.

[BCC+02] W. H. Bell, D. G. Cameron, L. Capozza, A. P. Millar, and al. Simulation of dynamic grid replication strategies in optorsim. In *Proceedings of the Third International Workshop on Grid Computing*, pages 46–57. Springer-Verlag, 2002.

[BDGL01] L. Becchetti, M. Draoli, C. Gaibisso, and F. Lombardi. A java implementation of a reliable multicast file transfer protocol: Design and evaluation. In *Proceedings of Internet and Multimedia Systems and*

Applications (IMSA), Honolulu, U.S.A., ISBN: 0-88986-299-0 (340), pages 163–168. ACTA Press (Calgary), August 2001.

[BGGL04] M. Bianchi, C. Gaibisso, G. Gambosi, and F. Lombardi. Smart dynamic selection of web service access points. In *Proceedings of Parallel and Distributed Computing and Systems (PDCS), Cambridge, U.S.A.*, ISBN: 0-88986-421-7 (439), pages 755–760. ACTA Press (Calgary), October 2004.

[BHG87] P.A. Bernstain, V. Hadzilacos, and N. Goodman. *Concurrency control and Recovery in Database Systems*. Addyson Wesley, 1987.

[BKTN98] S. Bhattacharyya, J. F. Kurose, D. F. Towsley, and R. Nagarajan. Efficient rate-controlled bulk data transfer using multiple multicast groups. In *IEEE INFOCOM (3)*, pages 1172–1179, 1998.

[Bos99] R. Bostrom. *A Study On the Performance of MFTP Over Satellite*. PhD thesis, Telia Research AB, 1999.

[CA03] Robert C. Chalmers and Kevin C. Almeroth. On the topology of multicast trees. *IEEE/ACM Trans. Netw.*, 11(1):153–165, 2003.

[Cas02] H. Casanova. Distributed computing research issues in grid computing. *ACM SIGACT News*, 33(3):50–70, 2002.

[CCSM+03] D. G. Cameron, R. Carvajal-Schiaffino, A. P. Millar, C. Nicholson, and al. Evaluating scheduling and replica optimisation strategies in optorsim. In *Proceedings of the Fourth International Workshop on Grid Computing*, page 52. IEEE Computer Society, 2003.

[CDF02] B. Cain, S. Deering, and B. Fenner. Internet group management protocol, version 3. RFC 3376, 2002.

[CDKR02] M. Castro, P. Druschel, A. Kermarrec, and A. Rowstron. Scribe: a large-scale and decentralised application-level multicast infrastructure. *IEEE Journal on Selected Areas in Communications (JSAC) Special issue on Network Support for Multicast Communications*, 20(8), 2002.

[CGI+99] R. Canetti, J. Garay, G. Itkis, D.Micciancio, and al. Multicast security: A taxonomy and some efficient constructions. In *Proceedings IEEE INFOCOM*, volume 2, pages 708–716, 1999.

[CHKW98] D.M. Chiu, S. Hurst, M. Kadansky, and J. Wesley. Tram : A tree-based reliable multicast protocol - technical report tr-98-66, sun microsystems, 1998.

[CK00] S. Cho and C. Kim. A secure multicast architecture with a decentralized key management. In *Proceedings of the International Conference on Electronic Commerce, Seoul, Korea*, Aug 2000.

[CKM+03] J. Cui, J. Kim, D. Maggiorini, K. Boussetta, and al. Aggregated multicast - a comparative study. In *special issue of Cluster Computing: The Journal of Networks, Software and Applications, Baltzer Science Publisher*, pages 1032–1044, 2003.

[CL87] D. Clark and L. Lambert. Netblt: A bulk data transfer protocol. RFC 998 - http://www.rfc-archive.org, 1987.

[CP99] R. Canetti and B. Pinkas. A taxonomy of multicast security issues. Internet Draft - http://www.securemulticast.org/draft-irtf-smug-taxonomy-01.txt, 1999.

[CPW97] L. Cingiser, A. Di Pippo, and V. Wolfe. Object-based semantic real-time concurrency control with bounded imprecision. *IEEE Transactions on Knowledge and Data Engineering*, 9(1):135–147, 1997.

[Dat] EU DataGrid. The datagrid project - http://www.eu-datagrid.org.

[DAZ99] M.J. Donahoo, M. H. Ammar, and E. W. Zegura. Multiple-channel multicast scheduling for scalable bulk-data transport. In *IEEE INFOCOM*, pages 847–855, 1999.

[Dee89] S. Deering. Host extensions for ip multicasting. RFC 1112, August 1989.

[DGLS02a] M. Draoli, C. Gaibisso, F. Lombardi, and A. Stentella. A java architecture for secure reliable multicast data transfer: Performance evaluation. In *Proceedings of Communication Internet and Information Technology (CIIT), St. Thomas, U.S.A.*, ISBN: 0-88986-327-X (376), pages 142–147. ACTA Press (Calgary), November 2002.

[DGLS02b] M. Draoli, C. Gaibisso, F. Lombardi, and A. Stentella. Performance evaluation of a reliable and secure multicast bulk data transfer application in java. In *Proceedings of Computer Measurement Group Workshop (CMG), Rome, Italy*, June 2002.

[DR04] T. Dierks and E. Rescorla. The tls protocol v1.1. RFC2246 - http://www.ietf.org/internet-drafts/draft-ietf-tls-rfc2246-bis-08.txt, 2004.

[EKK+02] M. Ellert, A. Konstantinov, B. Konyac, O. Smirnovac, and al. Performance evaluation of the gridftp within the nordugrid project. E-print cs.DS/0205023. - http://arxiv.org/pdf/cs.DC/0205023, 2002.

[ES04] A. El-Sayed. *Application-Level Multicast Transmission Techniques Over The Internet*. PhD thesis, Institut National Politechnique de Grenoble, 2004.

[Est03] D. Estrin. Protocol independent multicast - sparse mode (pim-sm): Protocol specification (revised). internet draft http://netweb.usc.edu/pim/internet-drafts/draft-ietf-pim-sm-v2-new-08.txt, 2003.

[FJL+97] S. Floyd, V. Jacobson, C. Liu, S. McCanne, and al. A reliable multicast framework for light-weight sessions and application level framing. *IEEE/ACM Transactions on Networking*, 5(6):784–803, 1997.

[FKNT02] I. Foster, C. Kesselman, J. Nick, and S. Tuecke. The physiology of the grid: An open grid services architecture for distributed systems integration - http://www.globus.org/research/papers/ogsa.pdf, 2002.

[FKT01] I. Foster, C. Kesselman, and S. Tuecke. The anatomy of the Grid: Enabling scalable virtual organizations. *Lecture Notes in Computer Science*, 2150:1–10, 2001.

[For] Grid Forum. Global grid forum - http://www.gridforum.org.

[Foua] The Apache Software Foundation. http://ws.apache.org.

[Foub] The Apache Software Foundation. Axis. http://ws.apache.org/axis.

[Fouc] The Apache Software Foundation. Juddi. http://ws.apache.org/juddi.

[GKL+02] L. Guy, P. Kunszt, E. Laure, H. Stockinger, and al. Replica management in data grids - technical report, global grid forum (ggf5) working draft - http://citeseer.ist.psu.edu/guy02replica.html, 2002.

[GL05] C. Gaibisso and F. Lombardi. A reliable multicast approach to replica management for grids. In *Proceedings of Parallel and Distributed Computing and Networks (PDCN), Innsbruck, Austria*. ACTA Press (Calgary), February 2005.

[HAA99] J. Holliday, D. Agrawal, and A. E. Abbadi. The performance of database replication with group multicast. In *Proceedings of IEEE International Symposium on Fault Tolerant Computing (FTCS29)*, pages 158–165, 1999.

[HBH03] G. Hjalmtysson, B. Brynjulfsson, and O. R. Helgason. Overcoming last-hop/first-hop problems in ip multicast. In *Proceedings of Fifth International Workshop on Networked Group Communications (NGC/ICQT03) - Munich*, pages 205–213, September 2003.

[HJS+00] W. Hoschek, F.J. Janez, A. Samar, H. Stockinger, and al. Data management in an international data grid project. In *Proceedings of the First IEEE/ACM International Workshop on Grid Computing (GRID2000)*, volume 1971, pages 77–90. Springer-Verlag, 2000.

[HMCG04] H. Harney, U. Meth, A. Colegrove, and G. Gross. Group secure association key management protocol(gsakmp). Internet Draft - http://www.ietf.org/internet-drafts/draft-ietf-msec-gsakmp-sec-06.txt, 2004.

[HW04] T. Hardjono and B. Weis. The multicast group security architecture. RFC 3740 - http://www.faqs.org/rfcs/rfc3740.html, 2004.

[IGT04] R. Izmailov, S. Ganguly, and N. Tu. Fast parallel file replication in data grid. In *Future of Grid Data Environments workshop, Global Grid Forum (GGF10)*, 2004.

[JPOD96] J.Gray, P.Helland, O. O'Neil, and D.Shasha. The dangers of replication and a solution. In *Proceedings of ACM SIGMOD*, pages 173–182, Nov 1996.

[KHTK00] S. K. Kasera, G. Hjálmtýsson, D. F. Towsley, and J. F. Kurose. Scalable reliable multicast using multiple multicast channels. *IEEE/ACM Transactions on Networking*, 8(3):294–310, 2000.

[KKL04] G. Kola, T. Kosar, and M. Livny. Profiling grid data transfer protocols and servers. In *Proceedings of Euro-Par 2004*, volume 3149, pages 452–459, Pisa, Italy, September 2004. Springer.

[KLSS04] P. Kunszt, E. Laure, H. Stockinger, and K. Stockinger. Advanced replica management with reptor. *Lecture Notes in Computer Science*, 3019:848–855, 2004.

[Kre95] J. A. Kreibich. The mbone: the internet's other backbone. *ACM Crossroads*, 2(1):5–7, 1995.

[Kre01] H. Kreger. Web services conceptual architecture (wsca 1.0). http://www-3.ibm.com/software/solutions/webservices/pdf/WSCA.pdf, May 2001.

[Lam78] L. Lamport. Secure, efficient data transport and replica management. *IEEE Symposium on Mass Storage Systems*, 1978.

[LCG04] L. Lao, J. Cui, and M. Gerla. A scalable overlay multicast architecture for large-scale applications, 2004. UCLA CSD Technical Report no.040008, http:// www.cs.ucla.edu/NRL/hpi/papers/2004-tr-0.pdf.

[Lea01] D. Lea. *Concurrent Programming in Java. 2nd edition*. Addison Wesley, May 2001.

[LGLA98] B. N. Levine and J. J. Garcia-Luna-Aceves. A comparison of reliable multicast protocols. *ACM Multimedia Systems*, 6(5):334–348, 1998.

[LGT98] Li-Wei H. Lehman, S. J. Garland, and David L. Tennenhouse. Active reliable multicast. In *IEEE INFOCOM (2)*, pages 581–589, 1998.

[Lia98] T. Liao. Light-weight reliable multicast protocol specification. Internet Draft - http://webcanal.inria.fr/lrmp/draft-liao-lrmp-00.txt, 1998.

[LLGLA96] B. N. Levine, D. B. Lavo, and J. J. Garcia-Luna-Aceves. The case for reliable concurrent multicasting using shared ack trees. In *ACM Multimedia Systems*, pages 365–376, 1996.

[LP96] J. C. Lin and S. Paul. RMTP: A reliable multicast transport protocol. In *IEEE INFOCOM*, pages 1414–1424, March 1996.

[LP05] F. Lombardi and R. Puccinelli. Jet-lag java heterogeneous log analysis on the grid: architecture, implementation and performance evaluation. In *Proceedings of Parallel and Distributed Computing and Networks (PDCN), Innsbruck, Austria*. ACTA Press (Calgary), February 2005.

[LXC03] X. Liu, H. Xia, and A. Chien. Network emulation tools for modeling grid behavior. In *Proceedings of 3rd IEEE/ACM International Symposium on Cluster Computing and the Grid (CCGrid 2003)*, Tokyo, Japan, May 2003.

[Ma98] K. Miller and al. Starburst multicast file transfer protocol (mftp) specification. http://hegel.ittc.ukans.edu/ topics/internet/internet-drafts /draft-m/draft-miller-mftp-spec-03.txt, 1998.

[MBRES04] L. Mathy, N. Blundell, V. Roca, and A. El-Sayed. On cheats in application-level multicast. In *IEEE INFOCOM*, March 2004.

[Moy94] J. Moy. Multicast extensions to ospf. RFC 1584 - http://www.faqs.org/rfcs/rfc1584.html, 1994.

[MP02] M. Maimour and D. Pham. Dynamic replier active reliable multicast (dyram). In *Proceedings of the Seventh International Symposium on Computers and Communications (ISCC'02)*, page 275, IEEE Computer Society, 2002.

[PMM03] R. Di Pietro, L. V. Mancini, and A. Mei. A time driven methodology for key dimensioning in multicast communications. *In Proceedings 18th*

IFIP International Information Security Conference, pages 121–132, 2003.

[PSS98] P. Parnes, K. Synnes, and D. Schefstrom. Lightweight application level multicast tunneling using mtunnel. *Special issue of Journal of Computer Communications*, 21(15):1295–1301, April 1998.

[PTK94] S. Pingali, D. Towsley, and J. F. Kurose. A comparison of sender-initiated and receiver-initiated reliable multicast protocols. In *Proceedings of the 1994 ACM SIGMETRICS conference on Measurement and modeling of computer systems*, pages 221–230. ACM Press, 1994.

[Pus98] T. Pusateri. Distance vector multicast routing protocol. draft-ietf-idmrdvmrp-v3-06, March 1998.

[RGK96] M. Rabinovich, N. H. Gehani, and A. Kononov. Scalable update propagation in epidemic replicated databases. In *Proceedings of the 5th International Conference on Extending Database Technology*, pages 207–222. Springer-Verlag, 1996.

[Riz97] L. Rizzo. Dummynet: a simple approach to the evaluation of network protocols. *ACM Computer Communication Review*, 27(1):31–41, 1997.

[RKH98] P. Rosenzweig, M. Kadansky, and S. Hanna. The java reliable multicast service: A reliable multicast library, 1998. Sun Microsystems Laboratories - Boston Center for Networking, http://www.experimentalstuff.com/Technologies/JRMS.

[SGLA99] C. Shields and J. J. Garcia-Luna-Aceves. Khip: a scalable protocol for secure multicast routing. *ACM SIGCOMM Computer Communication Review*, 29(4):53–64, 1999.

[SL00] Y. Saito and H.M. Levy. Optimistic replication for internet data services. In *Proceedings of the 14th International Conference on Distributed Computing*, pages 297–314. Springer-Verlag, 2000.

[Sof] Sun Java Software. Java 2 platform api specs. http://java.sun.com/products/ jdk/1.2/docs/guide/misc/ threadPrimitiveDeprecation.html.

[SRT+98] S.Kumar, P. Radoslavov, D. Thaler, C. Alaettinolu, and al. The masc/bgmp architecture for inter-domain multicast routing. In *Proceedings of the ACM SIGCOMM conference on Applications, technologies, architectures, and protocols for computer communication*, pages 93–104. ACM Press, 1998.

[SSMD02] H. Stockinger, A. Samar, S. Mufzaffar, and F. Donno. Grid data mirroring package (gdmp). In *EDMS Journal of Scientific Programming - Special Issue devoted to Grid Computing 10(2)*, pages 121–133, 2002.

[SWW03] T. Su-Wei and G. Waters. Building low delay application layer multicast trees. In *Proceeding of 4th Annual PostGraduate Symposium: The Convergence of Telecommunications, Networking and Broadcasting*, pages 27–32. EPSRC, Liverpool John Moore University, June 2003.

[SZJ02] S. Setia, S. Zhu, and S. Jajodia. A comparitive performance analysis of reliable group rekey transport protocols for secure multicast. *Performance Evaluation*, 49(1-4):21–41, 2002.

[UDD01] UDDI.Org. Executive white paper. http://www.uddi.org /whitepapers.html, November 2001.

[UDD03] UDDI.Org. Uddi version 3.0.1 - uddi spec technical committee specification. http://uddi.org/pubs/uddi-v3.0.1-20031014.htm, October 2003.

[VS02] S. Vazhkudai and J. M. Schopf. Predicting sporadic grid data transfers. In *Proceedings of the 11th IEEE International Symposium on High Performance Distributed Computing (HPDC'02)*, page 188. IEEE Computer Society, 2002.

[VSF02] S. Vazhkudai, J. M. Schopf, and I. Foster. Predicting the performance of wide area data transfers. In *Proceedings of the 16th International Parallel and Distributed Processing Symposium*, page 270. IEEE Computer Society, 2002.

[WB84] G. Wuu and A. Bernstein. Efficient solutions to the replicated log and dictionary problems. In *Proceedings of the third annual ACM symposium on Principles of distributed computing*, pages 233–242. ACM Press, 1984.

[WGL00] C. K. Wong, M. Gouda, and S. Lam. Secure group communications using key graphs. *IEEE/ACM Transactions on Networking*, 8(1):16–30, 2000.

[WHA99] D. Wallner, E. Harder, and R. Agee. Key management for multicast:issues and architectures. RFC2627, June 1999.

[YGS95] R. Yavatkar, J. Griffoen, and M. Sudan. A reliable dissemination protocol for interactive collaborative applications. In *Proceedings of the third ACM international conference on Multimedia*, pages 333–344. ACM Press, 1995.

[Zap04] D. Zappala. Alternate path routing for multicast. *IEEE/ACM Transactions on Networking*, 12(1):30–43, 2004.