Active Network Support for Multicast Applications

María Calderón, Marifeli Sedano, Arturo Azcorra, and Cristian Alonso
UPM

Abstract

This article analyzes the necessity and feasibility of designing a protocol for active networks that supports multicasting applications with different characteristics in terms of data loss tolerance. The article begins with a presentation of the service elements required by multicast applications, and from this study a network service description is given. The advantages of providing this service over active networks are studied. The service description is then used as the set of requirements for the design of the RMANP protocol, which is capable of providing the service over active network technology. Then a prototype implementation of RMANP over the Active Node Transport System (ANTS) is presented, and some data for the evaluation of its performance is provided. Finally, the main conclusions are that active networks provide flexible support for the development of new network services, but further improvements in runtime efficiency are required.

Traditional teleservices have been divided into telecommunications services (e.g., telephone, TV, or radio) and data services (e.g., remote terminal, file transfer, or e-mail), each with different requirements in terms of transfer rate, data loss probability, transit delay, delay-jitter specifications, and relationships between session participants. These different requirements led to the design of network technologies tailored (down to the physical layer) for the specific service to be supported. Throughout the recent past, different technological advances in signal digitization, data compression, data processing, and data transmission have resulted in the design of new classes of teleservices. Mobility, portability, multipoint-to-multipoint, multipoint-to-point, guaranteed variable transfer rates, flow synchronization, and a continuous range of throughput, transit delay, and delay-jitter values are some examples of the requirements placed on the network by teleservices existing today. This situation is leading to the design of an integrated network based on technologies providing an "open" network service as a means to support a broad range of teleservices while being prepared to integrate new technologies in order to adapt to changing requirements. Much effort has been made in developing a packet- (and cell-) switched networking technology that provides multiple classes of guaranteed transfer rates, while at the same time providing low transit delay and delay-jitter. However, considerably less effort has been made in developing a networking technology that supports a broad range of functional requirements.

In this sense, active networks [1, 2] represent a major breakthrough since they introduce a technology that will provide an integrated open network, in terms of not only quality of service (QoS), but also functional behavior. Per-packet and per-session specific processing at each network node enables new network services to be deployed across the network as soon as the teleservices that require them are ready. However, this flexibility provided by active network technology should not lead to a situation in which each implementation, or each teleservice, uses its own specifically designed packets and associated procedures on the active network platform.

In this context, the purpose of this article is to define a network service appropriate for the requirements of multicast applications, design a protocol based on active networks that provides this service, and present an implementation prototype over the Active Network Transport System (ANTS) [3] used to validate the concept. The next section describes the attributes of the applications considered to share network requirements, and the service elements supporting these requirements. We then discuss the advantages of using active network technology, justifying the convenience and feasibility of designing an integrated network protocol. RMANP is presented, a protocol designed on active networks capable of providing the services defined in the section below. We cover the most relevant aspects of the implementation prototype carried out on ANTS and provide performance evaluation data. In the last section we describe the conclusions derived from this work.

A Description of Service Elements for Multicast Applications

We have analyzed a series of characteristics or attributes in order to identify the elements composing an adequate communications service for multicast applications. The characteristics chosen to determine the network requirements, and therefore the network service, are:

- *Number of senders and receivers* in the multicast application session.
- *Mix of reliable and unreliable* — This attribute indicates whether the multicast application needs to send both reliable and unreliable data in the same multicast application session.
- *Reliability time* — This is the tolerable delay from the expected arrival time of a packet. It is closely related to delay-jitter. Some applications process data with a certain lifetime, after which the information becomes irrelevant to them. Although this type of application would rather receive everything, they tolerate some probability of losing data. Examples are most real-time applications and those that send state updates and periodically resynchronize sending the full state.
- *Delay bound* — This is an upper limit on end-to-end data transfer delay imposed by applications.
- *Open/closed session* — If the session is closed the application indicates the identity of the participants. On the other hand, if the session is open the application does not indicate who the participants are going to be, and in principle allows any system to join the session.
- *Known receivers* — This attribute indicates whether the application needs to know the identity of its receivers. Some applications need to know their receivers, for example, the case where a single receiver failure is sufficiently important for the whole session to be interrupted.
- *Synchronized start* — Some applications require that all participants (sender(s) and receivers) be present at start time.
- *Ordering* — The most usual ordering types required by applications are sender-based and total.

In order to produce a network service description, we attempt to relate the different application attributes and generalize their implications. First, the number of receivers is dependent on the type of session. In open sessions the protocol should support a very large number (up to hundreds of thousands) of receivers, but in closed sessions the number might be much lower. Second, we consider that providing three types of per-packet reliability might cover most cases. These types are unreliable, time-constrained reliable, and reliable. Third, all applications have a certain delay bound, which should be preserved even for large and/or sparse groups. Finally, we have abstracted the requirements relating to open/closed session, known receivers, and synchronized start into three session types, described below.

In addition to the previous aspects there are some general service requirements needed by multicast applications. The absence of duplicated packets is generally desirable. A flow control mechanism that adapts to the effective network bandwidth is necessary in order to minimize packet losses. A related problem is that receivers can be heterogeneous regarding their processing capabilities and the characteristics (bandwidth and delay) of the path that connects them to the sender. The existence of just one slow receiver can slow down the communication to an extent that is unacceptable for the application.

Based on these considerations, we propose a generic multicast service, supplied by a service provider by means of a specific protocol, to support multicast applications with the following service elements.

Data Service — This element allows an application to choose between three types of reliability for each data packet sent: *unreliable, time-constrained reliable*, and *reliable*. In unreliable packet service, the service provider does not guarantee in any way the delivery of that packet. In time-constrained reliable service, the service provider guarantees that it will do its best (store the packet, request retransmissions, retransmit packets, etc.) in order to ensure that all receivers get that packet before the associated time expires. Finally, in reliable service the provider ensures that the packet will be delivered to all receivers. In this case, if the service provider is unable to deliver data to a specific receiver, it will notify the application of this fact.

Session Service — This service element provides primitives to establish, manage, and terminate sessions. These can be one of three modes: *open group, controlled open group*, and *closed group*. In an open group any receiver is allowed to join the session, and the application is not informed of the identity of receivers. In a controlled open group any receiver is allowed to join, and the service provider will notify the application on every receiver that joins or leaves the multicast session. In both of the previously mentioned session modes, if a receiver joins a session after it has been initiated, the service provider will not resend data previously sent. In a closed group the application indicates to the service provider the identity of the receivers that must join the session. If one or more of the authorized receivers does not join the session, the service provider will notify its identity(ies) to the application. The provider will also inform the application of any receiver that leaves the session. Packets sent in the open group should have a time-constrained reliability or no reliability at all. Packets sent in the controlled open or closed group can be of any of the three reliability types.

Flow Control Service — This service element will allow the source to adapt to the available end-to-end bandwidth of all receivers. Additionally this service element is in charge of identifying the receiver(s) that are slowing down the throughput of the session in case where this situation arises. This is done only in controlled open group or closed group service modes. In case this event occurs, the application will be notified of the identity of the slower receivers, and it may choose to request their expulsion or to slow down the flow of data to accommodate it to the speed these receivers support.

Advantages of Active Networks to Support the Service Elements

The traditional approach has consisted of implementing tailored solutions for each specific application at the application layer. Another approach has been provision of reliable multicast service by means of an end-to-end protocol [4, 5]. In this article we propose to provide the required service using active network support. The idea of a network that provides additional services which simplify the tasks needed to achieve reliable multicast has already been proposed in the past [6–8]. Benefits of using intermediate processing have already been quantitatively evaluated. Published simulation studies have shown that:

- If some intermediate points perform acknowledgment (ACK) aggregation and buffering, implosion problems are avoided and protocol throughput is increased [9].
- It is better to do local recovery from intermediate systems than from other end systems that participate in a multicast session [10].

With the introduction of active networks the development of complex enhanced networking services (e.g., to provide reliable multicast service) becomes feasible. In the following paragraphs we will prove that active networks can greatly simplify the provision of some of the service elements required by multicast applications.

The fact that *the number of receivers is high* causes two associated problems: ACK implosion and/or negative ACK (NACK) implosion. Focusing on the ACK implosion problem, active networks can perform *ACK fusing* at active nodes between the receivers and the source. ACK fusing consists of sending just one ACK from a given active node toward the source of each *n* ACKs received. The new ACK carries the fused information of all *n* ACKs. In respect to the NACK implosion problem, active networks can perform *NACK filtering* at active nodes between the receivers and the source. NACK filtering consists of the recording in active nodes of the NACK capsules (see [2] for the definition of capsule) already sent toward the sender. That is, they remember the data already requested, and when a NACK is received it is forwarded only if it asks for different data.

In relation to *reliability time* and *delay bound,* active networks offer two possibilities:

- They perform soft state caching of multicast data at some intermediate active nodes.
- They perform intermediate sequence control.

Data caching allows the implementation of a *local recovery* scheme that avoids all retransmissions having to be made from the sender. When an active node receives a NACK traveling toward the source, and has the requested multicast data in its cache, the node filters the NACK and retransmits the requested data. *Intermediate sequence control* consists of each active node controlling the multicast data it processes. When a sequence number gap is detected, a NACK capsule is generated toward the source.

Finally, the support of active networks in relation to the *knowledge of the identity of receivers* at the sender is based on an *aggregation function*. Receivers can indicate their identity by sending an ACK (or some other type of status) toward the sender. Given the possibility in active networks of carrying out the aggregation of these capsules, the sender can be aware of its receiver's identity without the problem of capsule implosion. Instead of processing one capsule from each receiver, the sender will process a reduced number of capsules, each with information related to many receivers.

Besides the specific requisites of multicast applications, in multicast communication protocols the objective of optimizing network resources is important, and active networks offer, in this aspect, the possibility of performing recovery with restricted scope and retransmission filtering. *Recovery with restricted scope* avoids the retransmission of a requested data capsule through all outgoing interfaces of a given active node. In this case, active network support consists of recording through which interfaces a NACK capsule for a given packet has arrived, and retransmission of that data capsule can be restricted to only these interfaces. Active nodes use *retransmission filtering* to prevent multiple retransmissions of the same capsule if it has been requested in parallel by a given set of receivers or active nodes which can be reached via the same network interface.

Reliable Multicast Active Network Protocol

The network service outlined above will be partially provided by the Reliable Multicast Active Network Protocol (RMANP), described below. It provides the data service and session service elements. The flow control service element is not supported. This protocol is designed for active network technology.

We assume that RMANP works on top of an unreliable network in which packets can be lost, duplicated, or reordered. RMANP relies on the existence of an active multicast routing service that handles receiver subscriptions from the group, and creates and maintains the multicast distribution tree. The current design of RMANP has the restriction that multicast routes from the source to the receivers, and unicast routes from the receivers to the source must be coincident (at least) at the active nodes. No assumption on the size of memory available at active nodes is made. RMANP is designed so that state information stored at active nodes can be flushed at any time; this is the "soft state" concept implied by [11, 12].

Capsule Types

Each capsule has an associated code to be executed in the active nodes when the capsules arrives at them. We have defined one type of capsule for each independent processing unit in the protocol. We have created different types of capsules for original and retransmitted data, even though they have the same format, in order to load the code needed to process retransmissions when the arrival of a NACK capsule occurs. We have defined an independent capsule for the unreliable data (UR capsule) because the only processing to be done with these data are replication and forwarding. We have the same capsule type (data capsule) for reliable data (RE flow) and time-constrained reliable data (T-RE flow) since their processing is very similar. The flow type is identified by the Flow field of the data capsule. The Flow field is also present in retransmission, ACK, multiple ACK (MACK), and NACK capsules. The term *direct descendants* of a given active node will be used to denote all receivers or active nodes that can be reached downstream from the given node without passing through other active nodes.

A brief explanation of most relevant capsule types follows.

UR Capsule

— This capsule carries data that the application wants to send unreliably.

Data and Retransmission Capsules

— These capsules contain data that the application wants to send reliably (RE flow) or reliably but restricted to a certain reliability time (T-RE flow). The Mack_Required field is used by the source to ask those active nodes that have receivers as their direct descendants to send a MACK capsule. The Data_RT field contains the reliability time to be used by active nodes and receivers in a T-RE flow.

ACK Capsule

— This capsule carries cumulative acknowledgments for RE or T-RE data.

MACK Capsule

— This capsule carries multiple accumulative acknowledgments. They are generated at the source's request by setting the Mack_Required flag in a data or retransmission capsule. This capsule carries the number of receivers that are direct descendants of the active node that generated this capsule, and for each receiver its address and highest sequence number (RE or T-RE flow) acknowledged.

NACK Capsule

— This capsule requests retransmission of RE or T-RE data. The Sequence_number field contains the base value used to calculate which capsules are requested. The Fields_number field indicates the number of Seq_Individ_NACK fields inside this capsule. Each Seq_Individ_NACK field is used to calculate the range of individual sequence numbers that are requested by the sender of this capsule. It has three subfields: Offset, Seq_n_NACK, and nack_c. The Offset subfield indicates the offset from the last sequence number of the preceding field. The Seq_n_NACK subfield contains the number of consecutive capsules requested (negatively acknowledged) starting with the next one indicated by the offset field. The nack_c subfield contains how

Sequence_number	Fields_number	Seq_Individ_NACK			Seq_Individ_NACK		
1419	2	0	13	3	7	1	2

Table 1. *A NACK capsule.*

many NACKs have already been sent for this range of sequence numbers.

The design of this capsule is based on the one described in [13]. In order to clarify this capsule format, Table 1 shows the NACK fields used to request the retransmission of capsules with sequence numbers from 1420 to 1432 for the third time, and capsule 1440 for the second time.

RRL Capsule — The rejected receivers list (RRL) capsule is multicast from the source and contains the addresses of the receivers to be rejected from the RMANP session.

Elements that Participate in an RMANP Session

RE and T-RE data capsules generated in the same session will be treated by RMANP as two data flows with independent sequence numbers and state information. The only difference between the processing of RE and T-RE data capsules is that the removal of a T-RE data capsule can be triggered by the expiration of its associated reliability time (Data_RT). From now on, all the processing will be explained for RE data capsules, and T-RE processing will only be highlighted when differences appear.

Source — The source stores RE and T-RE data capsules generated until the corresponding MACK capsules have been received with information from all receivers. In the case of T-RE data capsules, they are also removed when their Data_RT (the reliability time of the session) expires. The application is responsible for choosing the type of session service it requires for each RMANP session opened. The characteristics of the three session services are:

- *Open group* — Only UR and T-RE data capsules are used. In this kind of session the source does not have any information related to the receivers connected to the session.
- *Controlled open group* — All types of capsules (UR, RE, and T-RE data) are allowed in this session type. The source controls the receivers that are connected during the session by periodically requesting receiver status information. This request is done by setting the Mack_Required flag in RE or T-RE data or retransmission capsules. The source entity will inform the application about which receivers join or leave the session. If the application requests it, the source can promote the expulsion of any receiver by multicasting an RRL capsule.
- *Closed group* — The application indicates the list of receivers that should participate in the session. When the RMANP session is initiated, the source entity unicasts an invitation to each of those receivers. If one (or more) receiver(s) has not answered positively, the source entity will notify the application of this fact, which decides between aborting or continuing the session. Then, if new receivers connect they are automatically expulsed (using RRL capsule). In other aspects, this session type is similar to the controlled open group.

Receiver — Receivers send ACK capsules toward the source with the sequence numbers they have correctly received. ACK capsules are generated every certain number of data capsules received, or after a given ACK timer expires. Each time a capsule loss is detected the receiver unicasts a NACK capsule containing the range of lost sequence numbers to the source. When a loss occurs, a NACK timer is started (unless it is already running) in order to periodically generate a NACK capsule containing the sequence numbers (each range with its associated NACK count) of all those capsules whose retransmission is being expected. The NACK count (*nack_c* field) associated with each sequence number range is used to indicate how many times the NACK time has elapsed while waiting for this range. In the case of T-RE data capsules, the same process takes place, except that after their associated Data_RT has elapsed their retransmission is not requested anymore.

Active Node — Active nodes maintain state information for each multicast session (source and group) and for each of their direct descendants. Active nodes perform the following tasks.

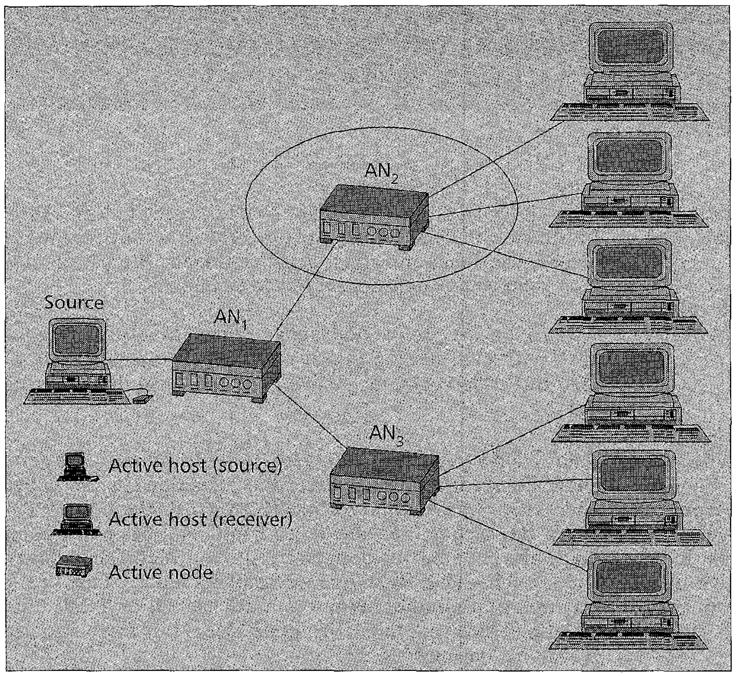
Data Caching and Buffer Release — In addition to the state information corresponding to its direct descendants, the active node keeps information for itself. It records the highest RE and T-RE accumulated sequence numbers it has buffered or that have been acknowledged by all its direct descendants, and the RE and T-RE sequence numbers buffered above the accumulated number. When a multicast RE or T-RE data capsule is received, the active node stores it if there is space available (*data caching*).

When a certain number of data capsules have been received, the active node unicasts an ACK capsule toward the source (*ACK fusing*) with the cumulative sequence number that has already been acknowledged by all its direct descendants or been stored by the active node itself. The generation of acknowledgments for capsules not yet acknowledged by all direct descendants, but stored in the active node, tries to avoid storing a given capsule in all active nodes that exist between a receiver and the source.

When an ACK capsule is received from one of its direct descendants, it is processed (in order to actualize the corresponding RE and T-RE sequence numbers of the direct descendants and of the active node itself) and is not forwarded. An ACK capsule has different meanings depending on whether it was generated by a receiver or an active node. If a receiver generates it, the ACK means that the receiver itself has received the capsule. If an active node generates the ACK, it means that either the active node has stored the corresponding data capsule, or all its direct descendants have acknowledged that capsule. Each data capsule is stored at an active node until all its direct descendants have acknowledged it, or the soft state storage time for that capsule has expired.

An active node will generate one MACK capsule toward the source upon the arrival of a data or retransmission capsule with the Mack_Required field set. Each MACK capsule carries the current acknowledgment information regarding all direct descendants of the active node that are receivers of the session (*aggregation function*). Only active nodes that have receivers as direct descendants will generate MACK capsules. MACK capsules are not processed at any intermediate active node as they travel to the source.

Retransmissions — Retransmissions in RMANP are based on the mechanisms proposed in [14]. Here we will use "interface" to denote an interface of the active nodes that leads to direct descendants. Active nodes use two type of records with state information that supports the retransmission process. One retransmission_info record is created for each retransmitted data capsule. This record holds a NACK count value (nack_c) for each network interface through which the capsule was retransmitted. It is used to filter unnecessary retransmissions that may occur when many receivers that can be accessed through a given interface do request retransmissions for a



■ Figure 1. *RMANP session topology.*

given capsule. One nack_supression_info record is created for each sequence number requested by the active node. It holds the nack_c value of the last NACK sent for this sequence number and a bitmap with as many entries as the number of interfaces of the active node. The bitmap indicates through which interfaces NACK capsules for this sequence number were received.

Active nodes or receivers start the retransmission process when they detect the loss of data capsules (*intermediate sequence control*), and they generate a NACK capsule with the requested sequence numbers and their associated nack_c. NACK capsules are processed at the first active node they visit on their way toward the source, and are not necessarily forwarded. When an active node receives a NACK capsule, it analyzes all the sequence numbers and nack_c fields contained in it. For each pair (sequence number, nack_c) the node carries out the following functions:

- It checks if the retransmission request for the sequence number has already been attended to (*retransmission filtering*) or not. If it has been attended, no more processing is done. Otherwise, the processing proceeds to the following point. To check if the retransmission has been attended to, it verifies that the *retransmision_info* record (associated with this sequence number and incoming interface) exists, and that the nack_c in the record is higher than or equal to the nack_c being processed.
- It checks if it has the requested data locally stored. In this case, it performs the retransmission (*local recovery*), and no more processing is done. Otherwise, the processing proceeds to the next point. The retransmission is performed by sending (in multicast) the data capsule only through the specific interface where the NACK capsule arrived (*recovery with restricted scope*) and actualizes its retransmission_info record to filter further retransmissions.
- It checks if a negative acknowledgment has not already been sent. In this case, a NACK has to be sent for sequence number and nack_c being processed, and it writes the nack_c being processed into the nack_c of the nack_supression_info record. Otherwise, no NACK has to be sent (*NACK filtering*). In either case, it has to set the bit of the nack_supression_info record corresponding to the interface through which this NACK was received. To check if the negative acknowledgment has already been sent, it verifies that the nack_supression_info record (associated to this sequence number) exists and that the nack_c in the record is higher than or equal to the nack_c being processed.

When the processing of all the sequence numbers is finished, it will generate a single (or no) NACK capsule to request all the sequence numbers needed.

When a retransmission capsule arrives at an active node, if the nack_supression_info record corresponding to the capsule exists it forwards the capsule only through the interfaces which are set in the associated bitmap (recovery with restricted scope). After that, it updates the retransmission_info record for that sequence number. If no nack_supression_info record is found (cache space is low, active node drops, etc.) the active node caches the data in the retransmission capsule. If no cache space is available, the capsule is forwarded in normal multicast mode and its associated retransmission_info record updated.

Management of Receivers/Active Nodes — RMANP does not need specific procedures to treat either distribution tree joins, leaves and modifications of receivers/nodes, or active node information losses. If a descendant of an active node leaves a given RMANP session, its associated state information at its active node ancestor will disappear when its time to live expires. When a new direct descendant joins the current session, ACKs and NACKs are received from it, and the active node creates the new necessary information accordingly. If, as a result of reorganizations of the distribution tree, an active node or receiver changes its relative position within the tree, the protocol will treat it as a session leave and subsequent session join.

Implementation of RMANP Over ANTS

The ANTS Platform

The Active Node Transport System (ANTS [3, 15]) is a Java-based toolkit for experimenting with active networks. It provides a node runtime and protocol programming model that allow users to customize the processing of their capsules. The main features of ANTS v. 1.2 are:

- Each capsule carries a protocol and capsule identifiers that point to the Java code required to process the capsule at each active node. The code is demand-loaded at active nodes when the first capsule of a given type arrives. Therefore, an active node will never load protocol code it does not need.
- In order to load the code, it is requested of the previous active node that processed the capsule. The objective is to receive the code from the closest possible place. The code is retained for some time at the node in order to reduce the number of requests for code-loading. All capsules of the same protocol and type share the same code at the node.
- Each active node has an LRU cache memory to implement a soft-state information repository. A time-to-live value is associated with each stored item. Items are removed when their associated time to live expires. In case there is not enough memory to store a new item, the oldest item is

automatically removed to free enough space to store the new one.

- ANTS uses UDP for active nodes to communicate with each other, and both the platform and capsule code are executed as a user-level process.

One advantage of being coded in Java is portability, but it has the drawback of reduced execution performance. Based on our work over the ANTS platform, we judge that it is a user-friendly and powerful support environment. However, we have introduced two modifications of ANTS that we have found useful for the development of RMANP. First, we have incorporated a method that allows checking at any given moment whether there is cache space available. This is useful in memory shortage situations in which it is better to discard a new item than to erase an old one. For example, in RMANP it is better to avoid caching a new data capsule instead of erasing protocol state information. Also, we have added a method that explicitly removes an item from the cache. In RMANP this allows memory used by cached data capsules that have been acknowledged to be freed without having to wait for their time to live.

RMANP capsule	Byte-code size (bytes)
UR capsule	2127
Data capsule	4242
ACK capsule	3115
MACK capsule	1983
Retransmission capsule	4708
NACK capsule	4730
Objects shared by capsules	4505
Total	25,410
TCP Linux object code (bytes)	42,392

■ Table 2. *Byte-code sizes of RMANP capsules and TCP object code.*

RMANP Prototype and Evaluation Testbed

An RMANP prototype has been implemented in Java using the primitives supplied by ANTS v. 1.2 (the code is available at http://escorpio.ls.fi.upm.es/RMANP.html). The current version implements the reliable delivery data service (RE flow) and the unreliable delivery data service (UR flow). The capsules that have been implemented are therefore those that are required for these two services (see the fourth section). Related capsules are bound in groups in order to reduce the delay caused by demand loading of code. The following three groups have been defined: (data, ACK, and MACK capsules), (retransmission and NACK capsules), and (UR capsule). The Java code has been byte-code compiled using Java Developers Kit (JDK) v. 1.1.3.

To test the performance of the system and to understand its limitations and strengths we have conducted an experiment running ANTS over the Java interpreter (Java Virtual Machine, JVM) of JDK v. 1.1.3. The active network topology of the RMANP session used is shown in Fig. 1.

In the experiment, the source sent one 3000-kbyte file using RE data capsules, each with 1024 bytes of data. The source injected RE data capsules into the active network at a rate of 20 capsules/s. UR capsules were sent, interleaved with RE data capsules at an average rate of 4 capsules/s. The processing measurements were made at the active node labeled AN_2. The AN_2 node ran alone on a Pentium MMX 166 MHz with 128 Mbytes of RAM under Linux. The other nodes of the experiment ran on different machines connected by 10 Mb/s Ethernet.

Memory Requirements

Our first performance study consisted of analyzing the memory requirements at the active nodes. We measured the memory required to store the code associated with RMANP capsules and the memory required to store the state information associated with the RMANP session. Table 2 shows the byte-code sizes for RMANP capsules. The table presents the three capsule groups, and also the byte-code size of auxiliary objects shared by the capsules. As a reference value, the TCP object code in a Linux kernel v. 2.0.3 is 42,392 bytes.

Notice that each capsule group will only be loaded at an active node when required. For example, the retransmission capsule group will not be loaded at active nodes where no NACKs are being processed. These performance implications were taken into account during the design of RMANP (e.g., for the definition of two different capsules for data and retransmission, in spite of them both having the same fields).

The memory required at active nodes to keep the associated RMANP state information is dependent on the session topology. Each active node requires 49 bytes/RMANP session in which it is involved. In addition to that, 21 bytes are required for each of its direct descendants. Ten additional bytes are needed for each capsule lost, and each group of eight network interfaces used at the active node. Finally, storage for cached data and data in transit is required. This is not differential to any network-based retransmission technique, since the nodes will only be able to locally retransmit the data they are storing. Notice that the byte-code stored at the active nodes is shared by all RMANP sessions, and the state information values grow linearly (with low factors) with session size. Based on these results we consider that the memory requirements of the RMANP implementation over active networks are acceptable.

Capsule Processing Times

Our second performance study has consisted of analyzing the processing times needed to process the RMANP capsules in active nodes. We estimated the average processing time associated with capsules by recording the real-time values using a Java native method invoking the gettimeofday() system call. Table 3 shows the average real time used at an active node for processing the different capsule types. Notice that the CPU time must be below these figures. Notice also that these figures do not include all the processing carried out on the capsule, because the OS and the ANTS platform perform standard processing over each incoming capsule before invoking the specific code required to process it.

These processing times are considered unacceptably high in relation to the processing power of the supporting machine. Still, we judge the results very promising because the execution performance can be improved in different ways. In addition to

RMANP capsule type	Processing time (ms)
UR capsule	0.946
Data capsule (not generating a MACK capsule)	1.227
Data capsule (generating a MACK capsule)	1.593
ACK capsule	0.284
NACK capsule	0.757
Retransmission capsule	1.175
MACK capsule	0.259

■ Table 3. *Processing times of RMANP capsules.*

fine-tuning the implementation, an improvement can be obtained by machine-code compilation of active networking platforms. Since capsule code cannot be compiled (one might not expect all active nodes to be homogeneous in terms of OS and hardware), improvements in Java interpretation and/or Java fast compilation are required for acceptable performance. Performance improvements can also obtained by introducing in the architecture of active networks standard fields in the capsule header to request processing shortcuts. This suggestion is similar to the idea of hop-by-hop vs. end-to-end options in IPv6. As an application example, the MACK capsules of RMANP only require processing at the destination. If they could be tagged for just forwarding at intermediate nodes it would result in overall performance improvements.

Finally, we consider the combination of two approaches to capsule code distribution, demand-loading and capsule-carrying, to be convenient. Demand-loading is appropriate for the protocol code modules which must be available at the node throughout the session, and these modules are reused by sessions of different users of this specific protocol. Capsule-carrying is convenient for seldom used procedures or capsule types, in which the delay caused by demand-loading is not worth the saving in bandwidth and reusability. Although not explicitly addressed in the experiment, an example in RMANP is the code associated with session establishment and release. This code could be carried only in the capsules used for session establishment and release, not demand-loaded, because it is not required throughout the session.

Conclusion

We believe that our definition of a multicast network service that supports different classes of packet loss tolerances will simplify and accelerate the development of new multicast applications and teleservices. It has been shown that the design of a protocol which provides this service is feasible: RMANP provides a range of multicast session and data transfer services to its user applications, while keeping protocol complexity reasonably low.

Based on our work on the ANTS platform, we judge that it is a user-friendly and powerful support environment. We have also pointed out areas in which ANTS could be improved, such as its cache management features, capsule processing shortcuts, and support for protocol software structuring. The RMANP prototype has acceptable capsule code sizes and data-segment memory requirements. However, the figures for execution time are not acceptable. As described in the previous section, we do not believe that this is caused by an intrinsic drawback of either RMANP or active networking. Future versions of active networking platforms should take into account that runtime performance is an important requirement in order to support high data rates and/or a large number of protocol instances in an active node.

In relation to active network technology itself we believe that, based on the work described in this article, it provides the capability needed for fast development and deployment of sophisticated network protocols and services. The case study presented is considered a difficult problem to deal with; in spite of this, currently available active network support has proved quite valuable for the design and implementation of a prototype solution.

Acknowledgments

This work has been partly supported by CICYT (the Government Commission of Science and Technology) under project TIC97-0929 and by UPM (Universidad Politécnica de Madrid) under project A-9704.

References

[1] D. L. Tennenhouse and D. Wetherall, "Towards an Active Network Architecture," *Proc. Multimedia Comp. and Networking 96, MMCN '96*, San Jose, CA, Jan. 1996.
[2] D. L. Tennenhouse *et al.*, "A Survey of Active Network Research," *IEEE Commun. Mag.*, Jan. 1997, pp. 80–86.
[3] D. Wetherall, J. Guttag, and D. L. Tennenhouse, "ANTS: A Toolkit for Building and Dynamically Deploying Network Protocols," *Proc. IEEE OPENARCH '98*, San Francisco, CA, Apr. 1998.
[4] C. Diot, W. Dabbous, and J. Crowcroft, "Multipoint Communication: A Survey of Protocols, Functions, and Mechanisms," *IEEE JSAC*, vol. 15, no. 3, Apr. 1997, pp. 277–90.
[5] B. Levine and J. J. García-Luna Aceves, "A Comparison of Reliable Multicast Protocols," *Multimedia Sys.*, ACM/Springer, 1998.
[6] A. Azcorra and M. Calderón, "A Network-Based Approach to Reliable Multicast," *Proc. 2nd Wksp. Protocols for Multimedia Sys. PROMS '95*, Salzburg, Austria, Oct. 1995, pp. 393–404.
[7] B. Levine and J. J. García-Luna Aceves, "Improving Internet Multicast with Routing Labels," *Proc. IEEE ICNP '97*, Atlanta, GA, Oct. 1997, pp. 28–31.
[8] C. Papadopoulos, G. Parulkar, and G. Varghese, "An Error Control Scheme for Large-Scale Multicast Applications," *Proc. INFOCOM '98*, San Francisco, CA, Mar. 1998.
[9] M. Calderón, "Unificación de los protocolos de multipunto fiable optimizando la escalabilidad y el retardo," Ph.D. thesis, Facultad de Informática, UPM, Oct. 1996.
[10] A. Azcorra, M. Calderón, and M. Sedano, "A Strategy for Comparing Reliable Multicast Protocols Applied to RMNP and CTES," *Proc. IEEE Conf. Protocols for Multimedia Sys.-Multimedia Networking '97*, Santiago, Chile, Nov. 24–27, 1997.
[11] R. W. Watson, "Timer-Based Mechanisms in Reliable Transport Protocol Connection Management," *Comp. Networks*, no. 5, 1981, pp. 47–56.
[12] D. Clark, "The design Philosophy of the DARPA Internet Protocols," *Proc. ACM SIGCOMM '88*, Aug. 1988.
[13] M. Hofmann, "Enabling Group Communication in Global Networks," *Proc. Global Networking '97*, Calgary, Alberta, Canada, June 1997.
[14] L. Lehman, S. J. Garland, and D. L. Tennenhouse, "Active Reliable Multicast," *Proc. IEEE INFOCOM '98*, San Francisco, CA, Mar. 1998.
[15] D. Wetherall, "Developing Network Protocols with the ANTS Toolkit," Aug. 1997.

Biographies

MARIA CALDERON (mcalderon@fi.upm.es) received B.S. and M.S. degrees in informatics from U. Politécnica de Madrid (UPM), Spain, in 1991 and a Ph.D. degree from the same university in 1996. She is currently an associate professor in the Department of Languages, Systems and Software Engineering, U. Politécnica de Madrid, Spain. Her research interests include network protocols, multicast applications, and active networks.

MARIFELI SEDANO received B.S. and M.S. degrees in informatics from U. de Deusto in 1987. She is currently a doctoral research student in the Department of Languages, Systems and Software Engineering, UPM. She is investigating multicast protocols and congestion control.

ARTURO AZCORRA [M] (azcorra@dit.upm.es) received B.S. and M.S. degrees in telecommunications from UPM in 1986 and a Ph.D. degree from the same university in 1989. He is currently an associate professor in the Department of Telematics Engineering, UPM. Current research projects include broadband networks, multicast teleservices, intelligent agents, and IP/ATM convergence.

CRISTIAN ALONSO received a B.S. degree in informatics from UPM in 1995. He is currently a software engineer at Teldat Co. His research interest include network protocols, network management, and object-oriented programming.