A Naming Approach for ALF Design

Michael Fuchs[1,2], Christophe Diot[1], Thierry Turletti[1] and Markus Hofmann[2]

[1] INRIA Sophia Antipolis, 2004 route des Lucioles, 06902 Sophia Antipolis (France)

[2] Institute of Telematics, University of Karlsruhe (Germany)

Abstract

Traditional error protocols use sequence numbers to identify so-called protocol data units (PDUs). As long as the reliability of data transmission is the responsibility of a transport-level protocol, sequence numbers work well. A new generation of protocols based on Application Level Framing (ALF) *[3] place the control of data transmission, i.e. managing errors and congestion, at the application level. However, sequence numbers do not allow ALF protocols to take advantage of their control capabilities as sequence numbers have no semantic interpretation of the packet content. Consequently, naming of* Application Data Units (ADUs) *is needed to allow ALF applications to have greater control over data transmissions. In this paper, we propose a new hierarchical data naming approach. We show how this approach allows more flexible control over reliability semantics that are meaningful to the applications. We show how this mechanism can be integrated into the* Reliable Multicast Framing Protocol (RMFP) *[4] and we evaluate our approach. Steps toward a naming architecture for the Internet conclude this paper.*

1 Introduction

In recent years, the *Mbone* [12] has been deployed to slowly introduce IP-multicast on the Internet. The Mbone is an overlay network that is constantly growing in size and covers a large subset of the Internet.

The availability of a multicast service in the Internet allows the widespread use of applications for group communication, such as conferencing tools [13], multicast file transfer [15], web-caching [19], distributed games [9], etc. One problem with multicast is the increasing algorithm complexity caused by various application requirements, especially in reliable multicast and congestion control protocols. Today, there is no *one-size-fits-all* protocol that can optimally serve the needs of all types of group applications. *Application Level Framing* (ALF) [3] has been proposed to help adapt transport-level services, such as reliability, to the needs of specific applications. ALF-based communication architectures are more flexible and efficient because they give the application more control over the end-to-end transmission. In particular, applications can optimize transmission control by using Application Data Units. ADUs are designed by the application to be data packets with a semantic interpretation. The goal of ADUs is to have data units independent of each others, that are immediately interpretable by the application. Such a semantic data organization in MPEG would be the organization of data following three types of ADU, each ADU containing a single I, B, or P frames, instead of having larger packets with no semantic signification.

Sequence numbers have been used in classic layered-architectures to control reliability. However, they do not allow ALF-based applications to take advantage of their control capabilities as sequence numbers have no semantic interpretation. Thus, a more sophisticated ADU identification mechanism, such as naming, is required. Naming allows the receiving applications to control the transmission of ADUs based on their semantics, rather than based solely on the position of bytes in the data flow.

ADU names offer a powerful mechanism for applications to decide what treatment to apply to an ADU type (sequence numbers are now used within each ADU type).

Naming is not only required by ALF. Some applications such as web browsers, or in the future shared virtual worlds and distributed interactive applications need a robust naming infrastructure. These applications are characterized by the manipulation of a large number of independent and dynamic objects that need to be uniquely identified in a session.

We present a new hierarchical naming. This architecture is justified by ALF. Our approach enhances the applications control over by using naming to introduce semantic information into ADU headers. In Section 2, ALF is reviewed, and naming is justified within the ALF context. Section 3 focuses on the relationship between data naming and transmission control. Section 4 presents our naming approach. We also describe how to use naming for receiver-based semantic reliability. Semantic reliability allows receiver applications greater control over the requests for retransmissions. The integration of our naming approach within the Reliable Multicast Framing Protocol (RMFP), is shown. Section 5 discusses the limits of our approach. Naming creates an overhead that can grow with the number of objects transferred. Extra processing and bandwidth will be required to advertise naming information. Section 6 compares our approach to related work, and in particular to the *Generalized Data Naming* architecture proposed by Raman and McCanne [16]. Section 7 concludes this paper with a discussion on the design of a naming architecture for the Internet .

2 Application Level Framing

Application Level Framing (ALF) is an architectural design principle for data communication protocols introduced by Clark and Tennenhouse in [3].

Most protocol architectures today rely on a layered network stack: each layer has a specific task and is independent from the other layers. Accordingly, each layer has no knowledge of the content of the information it processes. Data to be processed is received through a service interface that does not carry information about the data semantic. Transmission is controlled by a sequence number that is allocated by the sending protocol to detect losses. This sequence number cannot be used to identify the nature of the missing packet, and its relative importance.

One consequence of layer independence is that data must be *encapsulated* by each layer in a protocol data unit (PDU) before being passed to the next layer. Each layer can also *fragment* the data it has to transmit into several parts, and transmit each part as a separate PDU. Both encapsulation and fragmentation are transparent to the neighboring layers.

ALF violates the traditional layered architecture approach by integrating the end-to-end transmission control layers (4 to 7 in the OSI model), seeking a more flexible and functional decomposition. ALF proposes that data should be framed once at the application *level* into so-called ADUs, which consequently serve as the sole unit of transmission. Fragmentation, which is generally performed to cut data units to a size suitable for the network layer, is no longer possible, and it is up to the application to design ADUs with a size that is adequate for network transmission[1].

The ADU header is visible at all levels of the transmission path, from the application to the network. The encapsulation principle is no longer used, and each field can be utilized by any transmission control mechanism, although it must be clear which functional module is responsible for each field.

The goal of ALF is to provide flexible and efficient use of the network. In other words, ALF allows the application to be network conscious, and to adapt its behavior to the dynamics of the network.

It is not a coincidence if most of the multimedia applications available on the Internet today have been designed following the ALF architecture principle. ALF makes it easier to control the real-time properties of the application data in a networking environment that is not dedicated to real-time.

2.1 Out-of-sequence processing

Traditional transport protocols provide ordered delivery of data. This is mainly due to the layered architecture, to fragmentation, and to that historically, the first network applications could not process packets out-of-sequence (e.g. file transfers). Ordered transmission results in a bad utilization of the receiving entities resources when PDU loss occurs [8]. This is for a number of reasons:

- Received PDUs have to be buffered while waiting for the missing PDU to be retransmitted. PDUs can be delivered to the upper layer only when they are available in the correct sequence order. This leads the upper layer (and the CPU) to alternate between idle cycles and intense processing.
- Buffering at intermediate layers dramatically increases the time before data can be delivered to the application. Both delay and jitter consequently suffer ordered data transmission.
- Some applications can deliver partial information that can be useful to the user (e.g. in a web browser, or in a video server). This is not possible with ordered delivery.

In ALF, where ADUs are the sole unit of processing, transmission, and control, there is no more reason to buffer at intermediate steps. ADUs can be processed

[1] Note that in the current versions of IP, a new procedure called "MTU path discovery" has been implemented in order not to fragment IP packets during transmission.

independently of each other and out of sequence. Out-of-sequence processing has been shown to increase the performance of time-constrained applications, allowing for better CPU and memory utilization, shorter delays, and minimum jitter [8]. However, applications that rely on out-of-sequence data transmission have to perform ordering themselves (by buffering out-of-sequence ADUs). But not all types of applications require ordered processing and delivery of ADUs, and when needed, ADUs are only buffered once at the application level.

2.2 Treatment of missing ADUs

Transport protocols such as TCP buffer PDUs sent in a *retransmission buffer* in case packets are lost in the network. Sources then retransmit data based on either the reception of negative feedback from receivers, or after the expiration of a timer while waiting for positive feedback from receivers. Such an algorithm does not allow the application to decide what PDUs needs to be retransmitted; most applications would benefit from a finer grain retransmission control. Some may be able to easily retrieve missing data, while other applications may choose to send different data to repair missing information. Finally, real-time applications might even prefer no retransmissions at all for over-delayed data.

ALF gives applications control over ADU retransmission. Since the application inherently knows about the contents of ADUs, it can choose the most optimal way to repair missing information. Although this clearly increases the complexity of the application, it can minimize the sender's CPU and buffer load, and reduce the required network bandwidth.

This mechanism has been identified as ***semantic reliability*** [2], because the decision of how to treat missing ADUs depends on the semantic of the missing data, and not on the order of sequence.

Note that in ALF, packet numbering does not disappear. It is simply related to each type of ADU instead of being a unique sequence of numbers for the whole session.

3 Data Naming and transmission control

We have shown that the ALF approach makes networked applications more efficient, given that applications know how to define ADUs, and that they know how to process these ADUs independently of each other. Such an ADU-tailored transmission is only possible if the application can identify each ADU and associate with it a specific processing. In other words, an application-level context is needed to process ADUs efficiently. This context can only be provided by the source of the ADU. In a classic protocol architecture, the contextual information is limited to the sequence number. ALF suggests to use a special field in each ADU, the ***ADU name***, which provides contextual information required by the receiving entity to identify the correct processing for a received ADU. Contextual information can be the ADU name itself, or computable from the ADU name.

This section focuses on how ADU names can be used to improve transmission control.

3.1 Error control

Generally, there are two classes of reliable protocols for managing packet loss:

- Automatic Repeat Request (ARQ) protocols use control packets from the receivers to request the retransmission of lost data.
- Forward Error Control (FEC) protocols rely on the sender to encode redundant packets that can repair a limited amount of packet loss. FEC can reduce the loss-rate, but cannot guarantee 100 % reliable receipt of all data packets.

3.1.1 ARQ

Error control in ARQ consists of two steps:

1. **Loss detection:** protocol mechanisms detect the loss of data packets by gaps in the sequence number space, and, accordingly, a retransmission is requested.
2. **Loss recovery:** Losses are repaired by retransmissions of the lost packets. The receivers send control information to the sender (or to other group members in a multicast group) to request retransmissions.

ALF allows for a more efficient detection and recovery of missing packets as the decision to request a retransmission can be managed based on the semantic of the lost information (see Section 2.2).

A hierarchical ADU structure could improve more error control in both ARQ, allowing applications to replace a burst of missing ADUs by a higher abstraction information contained in less ADUs (or eventually in a single one).

3.1.2 FEC

FEC-based error control uses sequence numbers to detect losses. If a packet is lost, the protocol mechanisms attempt to reconstruct the lost packet from redundant information in others ADUs. This mechanism needs no feedback information from the receivers, but can lead to packet losses that can not be repaired.

Understanding packet semantics can help the receiving entity to select the appropriate redundant information and to discard (or eventually reject) part of the redundant information that is not needed. An other usage of the semantic information would be to discard systematically redundant ADUs received when no loss has been experienced in the previously received ADUs with the same name.

3.2 Flow and congestion control

Flow and congestion control protocols limit receiver and network congestion, respectively. ADU naming, can help in flow and congestion control [1].

- Filtering at the receiving entity (e.g. discarding ADUs that cannot be processed due to lack of resource based on their name) can solve the flow control problem. ADUs do not have to be transmitted to the application. Because of ADUs semantic interpretation, a receiving entity can decide to discard ADUs based on their the ADU name[2].

- Filtering at node level (still based on ADU names) can also help in congestion and flow control. In that case, ADUs are filtered within the network, because of local congestion, or because the receiving entities beyond this node are not interested in the filtered ADU.

3.3 Group management

It is often proposed in multimedia applications to organize the data transmission hierarchically, so that data of various importance (or with different signification) can be sent on different multicast groups. This approach is called layered multicast as each receiving entity can subscribe to the multicast group that carry data of interest only.

Layered multicast is more efficient if each multicast group carries a different type of ADUs, so that application can optimize the number data they process. The ADU name can mapped to the group address to make it easier to select what group to subscribe to.

4 Hierarchical naming with objects

This section describes our naming approach, which is based on hierarchical objects. The notion of *object* is defined, and the mapping of ADU names to objects is described. We also propose the use of Object Identifiers (OIDs) to reduce the overhead caused by the hierarchical ADU names. We conclude this section describing how our naming approach can be used to provide semantic reliability.

4.1 Hierarchical objects

Many applications do not use a single level of data structure, but rather a hierarchical data structure. For example, web browsers rely on URLs, which are complex data structure names. In a web site, each page can be reached by a hierarchy. In such a hierarchy, we define *objects* as being the node of the hierarchy. ADUs are defined as a semantic subdivision of objects which size is small enough to be carried independently on the network. ADUs can be found on intermediate nodes, as well as at the hierarchy leafs. For example, Figure 1 illustrates how the RM IRTF web page can be reached directly from the IRTF web. But it can also be reached by various other path, for example from the group chair page, or from the IETF home page. The IETF home page is an object, as well as any intermediate step in the path to the RM IRTF web. Once at the RM IRTF web, the hierarchical decomposition continues with different sections, on-line presentations, etc.

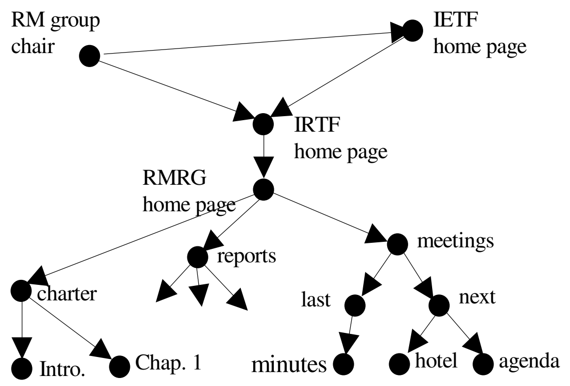

Figure 1 : Example of object hierarchy

The agenda of the next meeting can be identified with the name : *rm-irtf/next-meetings/next-meeting/agenda*. The RM IRTF can be named *with irtf/research-groups/ rm-irtf* or by *mark-handley/academic-page/research /groups/rm-irtf*. ADUs are numbered sequentially, and the minutes of the last meeting can be accesses by the sequence of ADUs: *[rm-irtf/last-meeting/minutes/#1; rm-irtf/last-meeting/minutes/#2; etc.]*.

In this object hierarchy, we propose the following definitions:

- An ***object name*** is the local identifier of the object. For example, *hotel*, *agenda*, or *charter* are object names.

[2] Note that this type of filtering can also be useful to the application when different types of ADUs are carried on the same multicast group.

- The ADU name is defined as the sequence of object names that leads to the object to which ADUs belong to, from the root of the hierarchy[3]. Consequently, *irtf/research-groups/rm-irtf* is a valid ADU name for the identification of the RM IRTF working group home page, as long as *irtf* is a unique object name (*www.irtf.com* for example).

Other examples of applications that require a naming infrastructure are newspaper distribution (very close to the previous example), or on-line games. Online games (or parent applications such as distributed battle fields or shared virtual environments) are the most complex examples. The hierarchy can be much more complex than with web applications, and also more dynamic, in that objects can be created and discarded dynamically.

4.2 Definition of the Object ID

We have identified two problems with this hierarchical naming approach :

- First, ADU names can be extremely long ; that does not make it easy or efficient to carry them in all ADUs.
- Second, there is more than one hierarchy that leads to an object.

We consequently need to map these ADU names on shorter identifiers. Each object will have a single identifier that will be unique for the session (even if this object's ADUs can be identified with different names). This identifier will also have to be shorter than the ADU names in order to be carried in each ADU.

The *Object ID* (*OID*) is this unique identifier that maps an object ADU name in order to solve the problems identified above.

This OID is carried in every ADU in the header, and the object to which an ADU belongs can be identified with this field. OID refers to the object, i.e. it does not contain the ADU sequence number. OIDs are unique in a session.

OIDs require a reduced bandwidth compared to ADU names. The ADU name can consequently be omitted in the ADU and only the OID is used to identify the object to which ADUs belong. OID is also independent of the path used to reach an object's ADUs.

The problem with OIDs is that the ADU name / OID mapping informations have to be advertized to all the interested entities in the session. The mapping information must be transmitted reliably, and as soon as there is a change in the hierarchy (e.g., new object creation, object mapping, etc.).

[3] Note that is root must be unique in the world, such as in a URL.

Several approaches are possible to advertize mapping informations:

- Mapping information can be transmitted in the first ADUs of each object (in the ADU name field). The drawback of this approach is that a new object's ADU can not be processed until one of the ADUs containing the mapping information is received successfully. The number of ADUs sent with the mapping information has to be chosen in order to guarantee that any entity will receive at least one of these initial ADUs.
- Mapping information can also be transmitted with a specific session announcement protocol. The session announcement packets have to be transmitted reliably, and before ADUs are exchanged between session entities. This protocol consequently introduces a delay in the object ADU transmission.

4.3 Objects and semantic reliability

OIDs are used for semantic reliability. Once an entity has received mapping information, it can decide whether an ADU can be discarded, must be delivered, or in the case of a missing ADU, if it has to be retransmitted, or where to find information to replace the missing ADU.

Although the definition of objects allows semantic reliability, it creates new problems for loss detection. Generally, transport protocols detect losses as gaps in the plain sequence number space of all ADUs. Plain sequencing means that ADUs are sequenced relative to the start of the session, and not relative to their object.

With our approach, the pair [OID, sequence number] is required to decide to which object a lost ADU belongs and to manage efficiently its retransmission. Accordingly, the OID is part of the sequencing information. The drawback of object related numbering is there is no ordering relation between the ADUs of different objects.

Due to the object nature (they can appear and disappear during the session lifetime) the following mechanisms are necessary to ensure correct loss detection and recovery:

- A *heartbeat* mechanism detects object tail-losses. Each sender use special control packets, so-called *heartbeats*, to transmit the highest sequence number of each object sent (every object has its own sequence number space). With this *heartbeat* information receivers can detect the loss of ADUs, even when no ADU with a higher sequence number has been received.
- Without some kind of garbage collection, the amount of information to indicate the highest sequence

number sent grows linearly with the number of objects sent. To limit the size of the heartbeat packets, the application has to explicitly *close* an object ADUs when transmission has stopped for that object. A *close* packet is transmitted containing similar information as the *heartbeat* packets. But the contained sequence number is indeed the sequence number of the last ADU of the object. Future *heartbeat* packets will not contain information about this object anymore. Note that this information has to be transmitted *reliably* to the session entities.

When a sender closes an object, that means that it has stopped sending new ADUs for this object. However, the ADUs of this object can still be requested for retransmission.

4.4 Application to RMFP

The *Reliable Multicast Framing Protocol (RMFP) [4]* is a framing protocol for reliable multicast transmission. RMFP has been designed on the same model as the *Real Time Protocol (RTP)* [17]. RMFP is not designed to be utilized at the transport layer of a conventional layered protocol architecture; instead it relies on an Application Level Framing architecture to provide more flexibility and efficiency to applications. RMFP does not specify any reliability mechanisms. It defines a set of packet formats (or encapsulation) that are intended to be used by all existing end-to-end reliable transmission protocols (such as SRM, RMTP, LGC, FEC, etc.). The reliable transmission protocol used by an application is identified as a RMFP profile.

The latest version of RMFP [4,7] describes the use of an OID in the ADU header. An implementation of RMFP with a SRM [6] protocol profile and with data naming has been developed. However, the mechanisms to provide semantic reliability are not yet integrated into SRM. This section presents changes necessary to provide semantic reliability within SRM. Since RMFP is a framing protocol, there are no changes necessary for RMFP packet formats. The new mechanism (*heartbeat* and *close* packets) affects only the reliability mechanisms of the protocol profiles.

It can be noticed that the *heartbeat* and *close* packets could have been defined to be part of the RMFP control protocol as it is the case with RTP and RTCP). We did not prefer this solution (even if it is more elegant and portable) because these two control packets requires a reliable transmission that is naturally provided by SRM.

4.4.1 The ADU

The ADU format is specified by RMFP and does not have to be changed. It already contains the necessary fields for semantic reliability: the OID, the sequence number, and the ADU name. The only change is the *use* of the OID and sequence number fields. The sequence number of an ADU is computed relative to the start of its object. The standard ADU format of RMFP is shown in figure 2.

0	8	16	24
Flags	Payload Tp.	Length	
Source ID			
Object ID		Sequence Number	
Name Len	ADU Name		
......			
DATA			
......			

Figure 2 : The RMFP ADU format

The ADU name field has a variable size. It can contain the ADU name or the object name (depending on the mapping information advertizing strategy). In most of the ADUs, it is the last field of the object name. This object name can be used as an object type identifier. If no session announcement protocol is used, the first ADUs of a new object carry the full ADU name in the ADU name field.

4.4.2 The SRM heartbeat packet

To indicate the highest-sequence-number-sent, the SRM profile uses the *heartbeat* packet. The object relative sequencing requires this packet to contain the highest sequence number sent for each object. The *close* packet has been designed to play the role of object garbage collector, so that the *heartbeat* packet size remains limited. The *close* packets have to be transmitted reliably, and therefore must contain a sequence number. Figure 3 shows the modified heartbeat packet for the SRM profile.

0	8	16	24
Flags	Payload Tp.	Length	
Source ID			
ST	Object Count	Close Sequence	
Object ID 1		Sequence Number 1	
......			
Object ID N		Sequence Number N	

Figure 3 : The heartbeat packet

The first eight bytes of the packet are the header used for all SRM control packets (for details see [7]). The *Subtype* field (*ST*) is used to identify the packet as a SRM *heartbeat* packet. The *Object Count* contains the number of [OIDs / sequence number] pairs present in the *heartbeat* packet. The *Close Sequence* is the sequence number of the latest close packet sent so far.

This field is used to detect the loss of the last control packet(s). The [OID / sequence number] pairs contain the OID and the highest sequence number sent for the objects currently transmitted.

4.4.3 The close packet

The *close* packet is introduced in SRM to provide the garbage collection for objects that are no longer used: to safely detect tail-losses of such closed objects, the last sequence number of those objects has to be transmitted reliably. For the objects closed, no *heartbeat* information is transmitted anymore. In this approach for RMFP-SRM the *close* packets are secured with the same reliability mechanisms used by SRM for ADUs. The *close* packets have their own sequence number space, similar to objects. Figure 4 shows the format of the new *close* packet.

0	8	16	24
Flags	Payload Tp.	Length	
Source ID			
ST	Object Count	Control Sequence	
Object ID 1		Sequence Number 1	
......			
Object ID N		Sequence Number N	

Figure 4 : The close packet

The common SRM packet header is followed by the *Subtype* field and the *Object Count*. The *Control Sequence* field is the sequence number of this *close* packet. It is used to detect and recover losses of close packets. The pairs of [OID / sequence number] indicate the sequence numbers of the last ADUs of the listed objects.

4.4.4 The SRM NACK packet

NACK packets are used to request retransmissions in SRM. To support the object relative sequencing, they have to include the OIDs of the requested ADUs additionally to their sequence numbers, see Figure 5.

0	8	16	24
Flags	Payload Tp.	Length	
Source ID			
ST	Object Count	Reserved	
Object ID 1		Sequence Number 1	
......			
Object ID N		Sequence Number N	

Figure 5 : The NACK packet

5 Discussion

Although the use of object names is intended to make transmission control easier and more efficient in networked applications, naming introduces its own set of problems. This section analyzes the drawbacks of our naming approach.

5.1 bandwidth overhead

The use of objects for semantic reliability was motivated by the idea of reducing the used bandwidth in avoiding unnecessary retransmissions. This is an optimization of *loss recovery*. The bandwidth overhead depends on the application's transmission characteristics.

The overhead created by the loss detection mechanism for objects has its origin mainly in new or more complex packets :

- Every time a protocol profile transmits sequencing information in a control packet, the OID has to be transmitted additionally to the sequence number. For packet types encoding several sequence numbers as *spans* (e.g. acknowledgment packets), this has the further drawback that each *span* has to be constrained to a single object.

- The *heartbeat* packets produce overhead that is proportional to the number of objects in transit. This overhead can be reduced by heuristics selecting the objects that contribute to the heartbeat packet, e.g. by means of the duration since the last ADU sent of each object.

- The overhead produced by the *close* packets is inversely proportional to the number of ADUs per object, since the close information has to be transmitted just once for every object.

The overhead introduced by object management is mainly dependent on two aspects: the number of ADUs per object, and the number of objects currently transmitted. Both parameters are application-dependent.

This overhead shows that applications that can not use semantic reliability work best with plain sequencing (i.e. only a single sequence number space is used for all ADUs instead of a sequence number space for each object). If semantic reliability is used, the amount of extra bandwidth is proportional to the number, to the size, and to the dynamic of application objects. A small number of objects reduces the use of *close* packets since the lifetime of the single is longer. *heartbeat* packets also contain a small number of [OID / sequence number] pairs.

5.2 Limitations of data naming in ALF

The mechanisms presented offer more flexibility to the applications, but also have performance drawbacks. This section focuses on three important issues: OIDs, retransmission control, and semantic reliability.

5.2.1 OID

OIDs must be unique within a session. Various techniques can be used to allocate OIDs, such as a hash function on the object name. Other labeling techniques based on the object hierarchy could also be used. See positional labels in [10] that could be used to mark the object hierarchy.

The size of the OID is also an open problem. In RMFP, it is arbitrary set to 2 bytes, which allows to identify up to 65536 objects. This is enough for experimental purpose. The OID size should be large enough to allow a simple coding of the OID, but small enough to minimize ADU size.

5.2.2 Retransmission control

Several problems with OID management can occur in case of multicast transmission:

- When data is issued by multiple servers; for example, in a distributed web-cache application, the original data is available at several locations. In that case, the OID has to identify the same object (despite different ADU names) for all servers. This complexifies in particular the OID construction.
- Another problem happens when reliable transmission with local recovery is used. The OID has to be sent to all the session entities, as well as related ADUs, in order to maximize retransmission efficiency (done by the closest entity that has received the missing ADU).
- When many ADUs are lost in the same object, some potential retransmitter might choose different strategies; a retransmitter may send all ADUs requested, resend the full object, or send a higher level description of the object. A robust transmission control protocol has to be used in order to control the retransmitters' behavior.
- Finally, the entity that requires the retransmission can use an object name that is not known by each participant, to directly request a high-level description of the missing object, or to request another object.

5.2.3 Semantic reliability

The problem of semantic reliability is that the application needs the OID and the sequence number of the lost ADU to decide how to react. The protocol, however, can only compute the sequence numbers of lost ADUs, since lost ADUs are detected by means of gaps in the sequence number space. The mapping information is mandatory for the application to choose the best retransmission strategy. It must be updated continuously and in real time. The loss of mapping information, or of control packets could be dramatic for the application that would momentarily be unable to detect and recover ADU losses.

6 Related Work

There have been various attempts to define naming structures for distributed applications. The most popular naming approach is the one used in URLs. Among other experiences we know about, we have selected:

- Levine and Garcia-Luna-Aceves [10] propose to add naming information to the multicast routing tree in order to complement the Internet group addresses (that are in fact group identifier) with semantic information. There three types of labels allow them to optimize transmission control and entity sub-grouping.
- TIBCO[18] uses subject-based naming for the TIB/RendezVous software. A subject name is a string describing the content of the message and specifying its destination. A subject-based naming scheme is required to handle dynamic programmable filters at the application level.
- Keryx [20] is another interesting approach where a naming approach and a dedicated abstract syntax (to identify object types) have been designed to support a distributed virtual environment.

McCanne and Raman [16] have proposed a General Data Naming scheme that is very similar to our approach. General Data Naming introduces hierarchically ordered *containers*. The leaf containers are similar to the objects in our approach, whereas the inner nodes of the container tree contain other containers. The *names* of the containers (i.e. ADU names in our approach) that are necessary for the application to process the ADUs are transmitted separately to the ADUs in *bind* packets. All leaf containers are independently sequenced. A *bind* session packet carries the mapping information of the *container descriptor* (equivalent to our OID) and the *container name*.

Both approaches face a common problem: the ADUs are sequenced relative to the start of their object, and, thus, the control information transmitted by the sender to enable the detection of tail-losses at the receivers has to provide the highest sequence number for every object. Our hierarchical object approach tries to limit the amount of this information by requiring the application to close completed objects. General Data

Naming allows an arbitrary number of containers within their container hierarchy.

Every container has a signature. For leaf containers consisting of ADUs, this is just the highest sequence number. For containers consisting of containers, the signature is calculated with a hash function out of the signatures of the child-containers. A special *Scaleable Session Announcement Protocol* (SSAP) is used between the sender and the receivers to detect the tail-losses: The sender periodically transmits an *update* packet that contains the signatures of the root-container and its child-containers. Every receiver calculates the signatures of all containers it has received, too. On receipt of the update packet it compares the signature in the update packet with the signature it has calculated for the root container in his view as receiver. If there is a discrepancy between the sender's and receiver's signatures of the root-container, the receiver compares the signatures of the child-containers. Since the root-containers signatures did not match, one or more of the child-containers signatures cannot match. The receiver sends a *query* packet, asking for the signatures of the child-containers with the discrepancies. The sender or another member with the requested information responds with a new update packet, this time with the signatures of the requested containers and their child-containers. This update/query cycle is repeated, until the signatures of leaf containers are transmitted in the update packets. These signatures consist of the sequence numbers of the last ADU sent for those containers. With this information, the receiver can finally request the retransmission for the lost ADUs.

The problem of this approach is the possibly long convergence time, until the receivers finally locate a loss. McCanne and Raman suggest to reduce this time using heuristics at the senders of the update packets to include the information that is probably required. We avoid this convergence problem using the two control packets (*heartbeat* and *close*). We have considered this approach to be more real-time, even if we have to pay the price with more bandwidth. Note that the control packets would be simple to implement in the General Data naming Scheme as part as the session announcement protocol (SSAP).

7 Summary

This paper presents an approach for a hierarchical data naming. The application can use *objects* as data units consisting of several ADUs. The common part of the ADU names of an object is identified as an object name, and is ideally transmitted only once per object. The ADUs carry only a short object ID field to determine the objects they belong to.

To efficiently implement semantic reliability, the ADUs are sequenced relative to the objects. This allows determination of which object a lost ADU belongs to. Thus, it is possible to decide the reliability requirement at the receiver for all lost ADUs.

However, the loss detection with object relative sequencing consumes more bandwidth in comparison to plain sequencing. This additional overhead is mainly dependent on the number of active objects (objects, for which more ADUs will be transmitted) and the number of ADUs per object. So, the object relative sequencing required by our naming approach should only be used for application that requires it:

- Semantic reliability can help some application to optimize retransmissions, and
- Distributed interactive applications such as web browsers, on-line games, and shared virtual worlds need such hierarchical naming to manage efficiently their data (made of numerous objects).

The hierarchical naming proposed in this paper, as well as the Generalized Data Naming are the first steps in the definition of a larger framework that would solve the naming/addressing problems for multipeer applications on the Internet.

The efficiency of the naming architecture is closely bound to addressing capabilities and to the association of objects and groups. Allocation of group addresses is a looming problem in the Internet; MASC [5] is a promising approach. But such general solutions do not give enough flexibility to complex application such as distributed interactive applications. Our future work involves:

- The implementation of our naming approach with a real application. This implementation will be based on the current RMFP/SRM implementation.
- The quantitative evaluation of the naming architecture in order to identify the real cost of the naming control overhead.
- The design of an object management group protocol (OGMP) to transform the current hierarchical naming approaches in a real naming/addressing framework. This protocol will aim to provide facilities for efficient object and group address management. OGMP should assist the application in object/group address mapping, in the OID construction, and in the naming control (implementing session announcement, *close* and *heartbeat* packets).

Acknowledgments

Thanks to Brian Levine and Jon Crowcroft for useful feedback on the paper, and on the related work.

References

[1] M. H. Ammar and L. R. Wu. *Improving the throughput of point-to-multipoint ARQ protocols through destination set splitting*. Proceedings of IEEE INFOCOM '92. Firenze. May 1992.

[2] P. Bagnall, R. Briscoe, and A. Poppitt. *Taxonomy of Communication Requirements for Large-scale Multicast Applications*. Internet-Draft <draft-taxonomy>, November 1997.

[3] D. Clark and D. Tennenhouse. *Architectural Considerations for a New Generation of Protocols.* In *Proceedings of ACM SIGCOMM '90*, pages 201-208. ACM, September 1990.

[4] J. Crowcroft, Z. Wang, A. Ghosh, M. Fuchs, C. Diot and T. Turletti. *RMFP: A Reliable Multicast Framing Protocol*. Internet-Draft <draft-crowcroft-rmfp-02.txt>, March 1998. See also the RMFP home page at URL www.inria.fr/rodeo/rmfp/.

[5] D. Estrin, M. Handley, S. Kumar, D. Thaler. *The Mulicast Address Set Claim (MASC) Protocol.* Internet-Draft <draft-ietf-idmr-masc-00.txt>. November 1997.

[6] S. Floyd, V. Jacobson, S. McCanne, C.-G. Liu, and L. Zhang. *A Reliable Multicast Framework for Light-weight Sessions and Application Level Framing*. IEEE/ACM Transactions on Networking, Vol. 5, No. 6, December 1997, pp. 784-803..

[7] M. Fuchs, C. Diot, T. Turletti, and M. Hofmann. *A Framework for Reliable Multicast in the Internet*, INRIA Research Report No. 3363, February 1998.

[8] F. Gagnon and C. Diot. *Impact of Out-of-Sequence Processing on Data Transmission Performance*. INRIA Research Report RR-3216. INRIA Sophia Antipolis (France). July 1997.

[9] L. Gautier, C. Diot. *Design and evaluation of MiMaze, a multiplayer game on the Internet.* Proceedings of IEEE Multimedia System, Austin, June 28 – July 1, 1998.

[10] B. N. Levine and J. J. Garcia-Luna-Aveces. *Improving Internet Multicast with Routing Labels.* Proceedings of ICNP '97. Atlanta. October 1997.

[11] C. Low, J. Randell, M. Wray. *Self-Describing Data Representation*. Internet-Draft <draft-low-sdr-00.txt>, October 1997.

[12] M. Macedonia, and D. Brutzman. *MBone provides Audio and Video across the Internet.* IEEE Comp. Magazine, pp. 30-36, April 1994.

[13] S. McCanne. *A distributed whiteboard for network conferencing.* UC Berkeley CS 268 Computer Networks term project May 1992, unpublished report; http://www.cs.berkeley.edu/~mccanne/papers/mccanne-wb92.ps.gz.

[14] J. Nonnemacher, G. Carle, M. Jung, M. Lacher, E. Biersack. *How Bad is reliable Multicast without Local Recovery ?* Proceedings of IEEE INFOCOM '98. San Francisco. April 1998.

[15] S. Paul, K. Sabnani, J.C. Lin, S. Bhattacharyya. *Reliable Multicast Transport Protocol (RMTP).* To appear in IEEE Journal on Selected Areas in Communications, April 1997.

[16] S. Raman and S. R. McCanne. *General Data Naming and Scalable State Announcements for Reliable Multicast*. Technical report, Computer Science Division (EECS), University of California, June 1997.

[17] H. Schulzrinne, S. Casner, R. Frederick, and V. Jacobson. *RTP : A Transport Protocol for Real-Time Applications.* RFC 1889, November 1995.

[18] TIBCO. *TIB/RendezVous White Paper*, available via URL http://www.tibco.com/ .

[19] D. Wessels, K. Claffy. *Internet Cache Protocol (ICP), version 2*. Internet Draft <draft-wessels-icp-v2-03.txt>, May 27, 1997.

[20] The KERYX web site. http://keryxsoft.hpl.hp.com.