1-1-1994

An Experiment In Modular Transport Protocol Design And Its Importance For Multicast Protocols

John Day
Umasankar Chandarlapaty
Margaret L. Loper

Find similar works at: https://stars.library.ucf.edu/istlibrary
University of Central Florida Libraries http://library.ucf.edu

This Research Report is brought to you for free and open access by the Digital Collections at STARS. It has been accepted for inclusion in Institute for Simulation and Training by an authorized administrator of STARS. For more information, please contact STARS@ucf.edu.

Recommended Citation
Day, John; Chandarlapaty, Umasankar; and Loper, Margaret L., "An Experiment In Modular Transport Protocol Design And Its Importance For Multicast Protocols" (1994). Institute for Simulation and Training. 12.
https://stars.library.ucf.edu/istlibrary/12
An Experiment in Modular Transport Protocol Design and its Importance for Multicast Protocols
An Experiment in Modular Transport Protocol Design and its Importance for Multicast Protocols
An Experiment in Modular Transport Protocol Design and its Importance for Multicast Protocols

IST-TR-94-09
February 25, 1994
Contract Number N61339-91-C-0103

Prepared by

John Day • Margaret Loper • Umasankar Chandarlapaty

Reviewed by

Darren Humphrey
An Experiment in Modular Transport Protocol Design and its Importance for Multicast Protocols

by
John Day
Margaret Loper
Umasankar Chandarlapaty

1. Introduction
This report describes the results of an experiment exploring new architectural directions in the design of communication protocols. Some researchers have attempted various "modular" approaches to protocol design in the past. However, these have been relatively narrow and have not been done in terms of a more comprehensive architectural framework and therefore have not been terribly successful. This report considers a modular approach that is part of a more comprehensive architecture developed by one of the authors, and explores the structural properties deriving from separating mechanism and policy in error control protocols, such as those found in the Transport Layer. First we will outline the architectural basis for this experiment,1 then we will describe in brief the basic protocol structure and the associated mechanisms and policies. We then turn our attention to the patterns in the structure of protocols that are made apparent by this approach and consider the implications of this work. The recognition of these unifying principles has also been important in clarifying some longstanding problems. For example, this work has shown that the solution of multicast transport is much more straightforward than previously understood and that the variety of forms of multicast transport can be much more easily accommodated than previously thought. But most importantly, this work shows that new transport protocols are not as important or necessary to solving many of the issues currently being considered as making the existing protocols more adaptable. Finally, we summarize our conclusions and indicate directions for future work.

2. The Architectural Basis
This experiment in exploring the structure of protocols is based on new architectural work that is currently under development. This work is based on the observation that there are a few key unifying principles in the architecture of communication protocols and that these can be used to distinct advantage to, among other things, increase the range of operation of a given protocol. In this section, we will briefly outline the more relevant of these unifying principles to the issues considered here.

1. This description touches on a small part of a more comprehensive framework and will be necessarily brief. A more detailed description of the architecture will be made in other forthcoming papers.
2.1. The Service of all Lower Layers is the Same

From the point of view of the user of a lower layer, the service provided can be summed up by the following:

The Transfer of Data among a set of users, identified by addresses, with a given Quality of Service.

In other words, the fundamental semantics of the lower layers are always the same. The major difference is only in the particular range of QoS provided by a layer. All of the lower layers have the same basic set of service primitives: open, close, send and receive. The only difference in the individual lower layers is the range of quality of service each one is able to support. Further, it can be shown that these primitives are not only necessary but also sufficient: no others need be included. In fact, including others can be detrimental to goals in the upper layers.

![Diagram of System A and System B](image)

**Figure 1.** In each system, there is subsystem for each layer that contains one or more protocol machines.

Every communication goes through three phases: enrollment, allocation, and data transfer. Enrollment makes the addresses known throughout the network, defines the characteristics of the communication that can occur, etc. Allocation\(^2\) establishes the required shared state to support the transfer of data and its associated mechanisms. Often the creation of this shared state is called establishing a connection. The allocation phase creates a virtual connection, flow, pipe, etc. among the protocol machines involved in the allocation phase. Data Transfer actually moves the data with the desired quality of service. The

---

\(^2\) Allocation is often called establishment. While the details of why a different term is used is beyond the scope of this paper, suffice it to say that it is more precise to view the operation as a request to *allocate* communication resources, rather than creating a pipe.
message or protocol-data-unit (PDU) to transfer user data is the only message that every protocol must have.

2.2. The Internal Structure of Layers

The lower layers can be modeled effectively in terms of a small number of module types that represent subsystems, embedded applications, data transfer protocol machines, mechanisms, and policies. These modules are combined in a limited number of ways to create the structures we find in the various layers. A subsystem is a collection of embedded applications and protocol machines in the same layer within one system. The subsystem moderates the local interactions among these embedded applications and protocol machines. A subsystem may have one or more types of protocol machines. Each protocol machine executes the procedures of a single protocol. A subsystem may have multiple instantiations of protocols of the same type. Each instantiation maintains the shared state of a single flow or connection. Embedded applications are found only in relaying layers and are used to manage the connectivity and resource allocation information of the relaying layer. Routing algorithms and their protocols are embedded applications.

The internal organization of embedded applications can be quite arbitrary and should be considered to be as general as that allowed by ISO 9545, the OSI Application Layer Structure, and may include application protocols with their own mechanisms and policies. Data transfer protocol machines contain one or more types of mechanisms. For each function, there are a small number of mechanisms that can provide the functionality. Mechanisms may have any of several policies. To illustrate, lost message detection is a function that some protocols require; acknowledgement is a mechanism that provides that function; and when or how often to acknowledge is a policy. (see below for more detail.)

2.3. The Separation of Mechanism and Policy

Each protocol is defined as a set of functions that achieve the basic requirements of that
protocol. The choice of functions is made based on the operating region in which the protocol is intended to exist and the desired level of service that is to result from its operation. Each protocol function consists of a mechanism along with one or more policies. Mechanisms are static and are not changed once the protocol is specified. Different policies are chosen for each connection when it is created. For any one mechanism, there are a variety of policies that may be applied to it. We mentioned lost message detection above as an example. As another example consider the flow control function, the basic sliding window is one flow control mechanism which is used in many protocols. Once implemented this mechanism is not modified. However, there are a variety of policies for determining when to extend credit and thus controlling the flow that may go with this mechanism: from simply extending new credit on receipt of a PDU, to periodically sending new credit, to high/low watermarks, to slow start, etc.

By separating policy and mechanism and allowing policy to be set at allocation time or to be modified during the Data Transfer phase, the operating range of a protocol can be increased and its ability to optimally serve a particular subset of the operating region can be greatly enhanced. The choice of policy depends on the traffic characteristics of the data being transferred, the Quality of Service (QoS) desired by the layer above and QoS provided by the layer below. Thus, the task of the protocol machine is to translate these QoS characteristics as supplied by the user of the protocol into a particular choice of mechanism and policy based on the QoS expected from the protocol machine in the layer below. Cleanly separating policy from mechanism is an important consideration in the design of a protocol that is capable of serving a wide range of QoS parameters while achieving an optimal operation. This also implies that QoS parameters should be defined based on their ability to select policy. While this has been known for some time, no lower layer protocols in use today have been defined to allow policy to be selected during the allocation phase. In today’s protocols, policy is specified in the specification and fixed at implementation.

2.4. The Concept of Context

For some time, the concept of context has been useful in the upper layers, i.e. Presentation Context and Application Context. It has been used, in some sense, to parameterize the protocol to select behavior not left open by the protocol specification. For example, the presentation protocol allows different concrete syntaxes to be selected for each distinct connection at establishment time. From this new architecture work, it has become clear that context is a general property of all protocols and is the means by which policy is negotiated.

All protocols should support the ability to negotiate policy at least during the allocation phase, if not during the Data Transfer phase as well. The negotiation of policy on a given connection is accomplished by defining a context parameter. The context is specified during the Allocation and/or Data Transfer Phases (see below). There are two basic approaches to defining the context for basic data transfer protocols: the most general approach would be based on a collection of QoS parameters alone. A more pragmatic approach might identify particular well-known operating regions based on traffic characteristics, mechanisms in a specific protocol, etc. and perhaps parameterize within these “well-known” regions.³

---

³ If there is any new facility that might be added to the lower layers as a result of this work, it would be the ability to alter context, i.e. policy, during the lifetime of an association.
2.5. The Two Kinds of Layers

There are two fundamental types of layers in the lower layers: error control layers, such as Transport and relaying layers, such as the Network Layer. The tendency is for these layers to alternate.

This alternating of functions is seen in traditional networks: The Data Link Layer provides “end-to-end” error control for relaying in the Physical Layer; The Transport Layer provides “end-to-end” error control for relaying in the Network Layer; One often resorts to “end-to-end” methods to ensure that relaying mail in the Application Layer has worked, etc. Although the separation is seldom completely clean in existing protocols, cleanly separating the relaying layers from the error control layers can allow considerable simplifications in implementation strategies and also provides an isolation of functionality. Exploring this separation is the primary purpose of this paper.

The application of policy also affects this alternating of layers. Policies in error control layers may be affected by the sender in indicating QoS, etc., but the policies are enforced by the receivers of PDUs. In relaying layers, policy is enforced by the senders. Given different traffic characteristics among the peer-protocol machines different policies may be useful among classes of peer-protocol machines participating in the same connection. In other words, all of the mechanisms in error control protocols are feedback mechanisms. There are no feedback mechanisms in the data transfer relaying protocols.

Note: This alternating layers on single associations (error control) with layers of functions on multiple associations (relaying) may appear to be an unnecessary constraint on the general mapping possibilities allowed by the modular structure referred to above. This is not so much a constraint, as a consequence of all layers having the same fundamental semantics: transfer data among a set of addresses. It is this fact that creates this underlying repeating structure among the modules, and thus in its purest form gives the appearance of a less rich structure. It is only an illusion that this is not the case for the upper layers. The upper layers exhibit the same fundamental structure. It is only the variety of additional functions that may occur in the data transfer phase of the upper layers that tends to obscure the similarity.

While there is nothing inherently wrong with two adjoining error control layers or two adjoining relaying layers, there are strong arguments against such configurations. Let Consider each case in turn. Two adjoining error control layers is fairly pointless since the scope of the two layers must be the same. Unless the first protocol is relatively weak, there should be no errors missed by the first one that will be detected or corrected by the second. If there are such errors, the second protocol should be used in place of the first. However in an existing network, it may not be possible to do anything about the existence or absence of the first protocol, in which case the second protocol may be necessary to achieve the desired QoS. This is probably the only instance where one should find two adjoining error control protocols.

---

4. It is recognized that most current transport protocols have in contradiction to this theory, a multiplexing function. Given that multiplexing is the only mechanism in the protocol with policy imposed at the source, we conclude that the “problem” is telling us that multiplexing should only occur in relaying layers. Subsequent work in developing these “patterns” has borne out this decision.

5. Since an error control layer bounds the scope of a relaying layer and there has been no intervening relaying layer that would increase the scope.
The argument against two adjoining relaying layers is less clear cut. It is the relaying layers that do multiplexing and routing while trying to avoid congestion. This is where PDUs are lost when congestion cannot be avoided. Two adjoining relaying layers would tend to compound the errors; thereby decreasing the QoS of the second relaying layer and thus impact the amount of QoS, as well as the performance, that the eventual error control layer could achieve. In addition, two adjoining relaying layers will usually (but not always) imply that the (N+1)-layer has wider scope than the (N)-layer. It is generally prudent (more efficient, less costly, etc.) to repair any errors in the layer with less scope, rather than propagating the errors to a (N+1)-layer with a wider scope and then attempting to recover from the errors. Thus, while it is not a hard requirement that there not be adjoining relaying layers, it will in most cases not be the best solution.

Thus, we should expect any relaying (N)-layer to have an error control layer as the (N+1)-layer. We will term this pairing a macro-layer. The error control layers tend to be less complex than the relaying layers. This is primarily because the mechanisms in error control layers apply to single associations. Whereas, the mechanisms in relaying layers are applied to multiple associations. Since there is no multiplexing in error control layers, address mapping between an (N)-layer which is an error control layer and an (N-1)- or (N+1)-layer which is a relaying layer is one-to-one. Error control layers tend to have mechanisms like: sequencing, flow control, acknowledgement schemes, data corruption detection, etc. While the relaying layers are more concerned with issues of resource allocation, such routing, concatenation, etc. Layer management protocols only occur in relaying layers. Thus, Relaying Layers are responsible for the mapping of addresses and routing, while the Error Control Layers are responsible for manipulation and maintenance of QoS. (The only exception to this is the choice of multiplexing strategies and specific allocations in the Relaying Layer can affect QoS.)

2.6. The Generic Protocol
For some time researchers have recognized that there was considerable similarity among protocols, especially error control protocols. It had become almost a standing joke about specifying the ultimate generic protocol. But from this, we can begin to see that the generic protocol is not a myth. There are actually a small number of mechanisms used in modern protocols. Most of the differences in the papers proposing "new" protocols are different policies, not different mechanisms.

The generic protocol consists of the basic data transfer mechanism to which various mechanisms and policies are added to achieve the desired range of quality of service. There is no generic protocol in the enrollment and allocation phases, since these may be accomplished in some cases without the exchange of protocol. The only fields required in the data transfer pdu are protocol id, protocol version, and protocol addressing information, i.e. either fully qualified addresses or shorter "connection-identifiers." By minimizing the functionality of the generic protocol, we can isolate each mechanism and consider its properties without influence from other mechanisms that might be in a complete protocol.

6. The one-to-one mapping of addresses between error control and relaying might tempt one to define the macro-layer as a layer. The fact that in practice it may be necessary to have two adjoining error control or relaying layers argues against over constraining the architecture by forcing this pairing.
3. An Experiment with the Transport Layer

3.1. The Methodology for the Experiment

The class of error control protocols generally viewed as transport protocols were chosen as the vehicle for this investigation. Transport protocols have most, if not all, of the functions generally found in lower layer protocols, such as flow control, lost and duplicate detection, sequencing, corrupted message detection, etc. Thus by considering the separation of mechanism and policy in transport protocols, one has considered much of the range of functions that might occur and thus have a good indication as to how such a separation would apply to protocols in general.

To test these architectural principles, we used a specification of a generic service and protocol. The generic service definition includes the allocation and data transfer phases to open, close, and send data. The generic protocol contains a data transfer pdu, assuming error free communication and no flow control. We then define the mechanisms for allocation, flow control, error control, etc. as separable modules that can be added to the generic protocol depending on what is required. Then one or more policies are defined with each of these mechanisms to consider the kinds of behavior that can be created by changing policy.

It should be understood that this is purely a gedanken experiment. We are not suggesting that this level of flexibility should be considered as an implementation strategy. The generic service and protocol are only specification vehicles for the purpose of this investigation. One would not expect implementations to attempt to negotiate mechanisms as well as policy “on the fly.” Mechanisms are chosen and specified in a protocol specification, although having a library of mechanism specifications can greatly simplify the specification process for specific protocols. An implementation is optimized to implement the particular selection of mechanisms in a specification.

To provide structure to the exercise, we adopted a state transition model that has been used with considerable success for several years. The specifications were structured around an outline that defined the conditions under which each service primitive and pdu type occurred and the action to be taken when it did occur. This form has been used successfully to specify services and protocols and produces well structured specifications that are easy to use and to implement. The form was extended to provide a structure for defining mechanisms and policies to be added to the generic protocol. (see the Appendix for the outline). This outline was then used to test how well mechanisms could be defined to be added to an arbitrary data transfer protocol. This methodology also provides a structure that makes it possible to deduce some properties of protocol design. In the next sections, we briefly provide an overview of the generic service, protocol, and the mechanisms that were defined. For details of the specifications, see the appendices.

3.2. The Generic Service

The generic service consists of allocate and deallocate service primitives to indicate that the layer should allocate the resources necessary to do data transfer and a data service primitive to transfer data in the data transfer phase.

3.3. The Generic Protocol

The generic protocol consists of a single data transfer pdu that carries addressing information. When an allocation primitive is received, the layer determines the allocation mecha-
nisms required to meet the quality of service requirements. Since the amount of shared
state required to support the desired QoS may or may not require the exchange of proto-
col, no protocol is part of the generic allocation protocol. All of the protocol for allocation
and deallocation is part of additional mechanisms. In most protocols, mechanisms will re-
quire additional information to carried to maintain the shared state. This information may
be carried on the data transfer pdu, or additional control pdus may be included in the pro-
tocol to carry this information. One of the major debates among protocol designers has
been which choice should be made and when. Some protocols have many control pdus; some put all control information in the data transfer pdu. As we will see below this ex-
periment clarifies the conditions for making this choice.

3.4. The Mechanisms and Policies
In this section, we briefly describe the mechanisms that were defined for the transport
protocol in this experiment. (For detailed specifications, see the appendices.) We have
not defined all of the mechanisms required for a fully functional protocol, for example
fragmentation/reassembly is not defined; but have concentrated on the major mechanisms
found in most transport protocols. These mechanisms cover the range of complexity from
simple to fairly complex and are sufficient for the purpose of this experiment to provide
considerable insight into what can be accomplished by not fixing policy as part of the pro-
tocol specification.

3.4.1. Allocation
We defined a single protocol mechanism for allocation/deallocation that can be used to
do either of the traditional allocation mechanisms: two-way or three-way handshake.
The allocation mechanism includes the means to specify the context, i.e the policies to
be used with this and other mechanisms included in this protocol. This mechanism
adds new control pdus to the protocol. The mechanism defines two sending policies
and a receiving policy.

3.4.2. Context Negotiation
While no protocols today provide for the negotiation of policy during the data transfer
phase based on our result that context is a general property of protocols, we have in-
cluded a basic mechanism to allow context to be changed during the data transfer
phase. This provides the general form of the mechanism found in the upper layers.

3.4.3. Sequencing
A sequencing mechanism has been defined that sequences on a pdu basis. This mecha-
nism adds new fields to the data transfer pdu header, but does not define any new con-
trol pdus. It defines a receiving policy.

3.4.4. Lost and Duplicate Detection
A mechanism defined for lost and duplicate detection, also called positive acknowl-
edgement. The mechanism uses the traditional dynamic width sliding window model
which requires the Sequencing mechanism. The mechanism defines an ack control
pdu. This control pdu is also defined to handle both positive and negative ack as well
as and selective ack and selective nack. Consequently, it requires six policies to char-
acterize the mechanism.

3.4.5. Flow Control
Similarly, the flow control mechanism was defined using the dynamic width sliding
window as a model, but we also allowed the same basic mechanism to define a rate
based flow control and used the same control pdu to change the rate. In fact, some in-
interesting flow control strategies can be created by using both mechanisms together. We defined a separate control pdu to carry the credits. This mechanism has two sending policies and one receiving policy.

3.4.6. Data Corruption
The data corruption mechanism allows different error codes to be defined as policy to allow the robustness of data corruption protection. The mechanism introduces a new field into the header of the data transfer pdu. No control pdu is necessary. There is one sending policy and one receiving policy.

3.5. Observations from the Experiment
There were a few mechanisms that had to have fields introduced into the header of the data transfer pdu: sequencing and data corruption. For these there were no control pdus required. However, it was found that the mechanisms could be made much more independent if we defined separate control pdus for the mechanisms where possible. For the mechanisms where this was possible there was no order of processing imposed by the mechanism itself. Most of the constraints in the ordering of mechanism execution that had been observed in other protocols was imposed by the policies, not the mechanism.

Protocols today do not provide the means to change policy once the connection has been established. We have provided such a mechanism. Further from an initial inspection of the kinds of policies that would be common it appears that many such changes can be made without adverse affects. This does not mean that there are no combinations of policies that do not create problems, but it does appear that a wide range of policy changes (and probably the ones most necessary for everyday operation) will not create pathological problems. This is a very interesting indication and requires further investigation.

There was one sending policy and one receiving policy for each mechanism. If there wasn’t, it was because we had overloaded the control pdu with multiple mechanisms. From a purist point of view, it might appear that this should be avoided but there are cases where it produces a reasonable simplification. For example, overloading ack and nack in simply accomplished by having a bit to distinguish which semantics the control pdu is conveying. On the other hand, we found that the coupling between acking and flow control in sliding window flow control which had been assumed to be an inherent part of this mechanism was all in the policy. Neither definition of flow control nor acknowledgement refers to the linkage between the edges of the window. Thus, it is prudent to define separate control pdus for acking and flow control to maintain as much independence as possible between these mechanisms. If a closer coupling is desired, it can be created in the associated policies. Although there is much to be said for maintaining a loose coupling among mechanisms. One should carefully consider the consequences of overloading a control pdu. In general, one should have a control pdu for each mechanism where the shared state does not have to be associated with the data transfer pdu. (Of course, this does not mean that one could not define a general control pdu to send whatever control information needed sending, essentially blocking the information together. However, this would complicate the processing of the control pdu, since it would not always contain the same fields.)
As a result of this exercise, we have been able to discern the following general principles in the structure of protocols.

4.1. We have been looking for the impossible. The first and probably most important observation was the one that, in some sense, started this work. A direct consequence of not separating mechanism from policy in the protocols we had designed up to now: Each of these transport protocols is made up of 5 or 6 mechanisms. By our continuing attempts to find “better” transport protocols and by not separating mechanisms from policy, we were saying that we believed that there was a single point in a 5 or 6 dimensional space that would satisfy all of our requirements! Once put this way the absurdity of our approach is clear. No reasonable engineer would ever expect to find such a point solution for such a wide range requirements.

4.2. Very few new mechanisms defined in last 20 years, but lots of policies. If one surveys the various protocols proposed for the data link and transport layers over the last 20 years, one finds that fundamentally there has not been any really new mechanisms proposed in some time. Most of these protocols are actually using different policies on a small set of mechanisms. Further, there are only a small number of mechanisms for each function. Basically, the variety of error control protocols can be described by a small number of mechanisms, variations in the lengths of the fields in the header, and a wide ranging collection of policies to go with the mechanisms.

4.3. There are two kinds of mechanisms: tightly coupled and loosely coupled.

4.3.1. Tightly bound mechanisms must be performed in a particular order and before any loosely coupled mechanisms. The information exchanged to maintain shared state must be associated with the data transfer pdu. Consequently, the elements of procedure are also associated with the data transfer pdu. Examples are sequencing and data corruption detection.

4.3.2. Loosely bound mechanisms may be performed in any order and shared state information exchanged is best transferred by distinct control pdus dedicated to that mechanism. Examples are lost and duplicate detection, acknowledgement, and flow control.

4.3.2.1. The elements of procedure are bound to the associated control pdu. In error control protocols, the sending policy is invoked upon receipt of a control pdu or the data transfer pdu, depending on whether the mechanism is loosely coupled or tightly coupled and the receiving policy upon receipt of the data transfer pdu or some internal event, e.g. a timer.

4.3.2.2. The number of PDU types in a protocol will be determined by: the data transfer pdu and one or more pdu types for each loosely coupled mechanism.

4.3.2.3. Other design considerations will determine whether or not these control pdus are mapped to the same (N-1)-address as the data transfer pdu, so-called “in-band signalling” or to different (N-1)-addresses, so-called “out-of-band signalling.”

4.4. In its purest form, a mechanism consists of three basic parts:

1) a procedure bound to a pdu: the data transfer pdu in the case of tightly coupled
mechanisms, a control pdu in the case of loosely coupled mechanisms;

2) a sending policy, invoked in the process of executing the procedure in 1).

3) a receiving policy, bound to the procedure associated with the data transfer pdu.

"Practicality" may lead to "overloading" a control pdu with multiple mechanisms. For example, ack/nack are different mechanisms, but adding one bit to the control pdu will allow the same pdu to be used for both mechanisms. However, these should be considered very carefully because the savings may be illusory; they also may be real.

4.5. Any coupling between mechanisms is done through policy. Specific policies may impose some ordering of the mechanisms. However, good policies will maintain the loose coupling to improve protocol processing efficiency. Shared state within a mechanism may be more closely coupled.

5. Implications of this Work


Data Link Protocols are specialized error control protocols. Because data link protocols are operating much closer to the media, it is important that they be specialized to accommodate the characteristics of the media. Designing and developing a new data link protocol can be greatly simplified and made more cost effective as a result of this work. To specify a new data link protocol given the characteristics of the media, one would simply choose the necessary mechanisms, choose appropriate widths for the fields, and the appropriate policies given the characteristics of the media. For a transport protocol one might want to build a fairly general implementation to accommodate the wide range of policies, while for a data link protocol one could build a much less general implementation because there would be fewer policies specific to the characteristics of the media to be accommodated. It is important that while the separation of mechanism and policy allows for very general protocols in the transport layer, it also makes the task of designing and developing highly specific data link protocols easier and much cheaper.

5.2. Multicast

One of the most important results of this work has been the implications for multicast transport. For some time, the primary problems in the development of multicast transport have been the myriad of forms "reliable" could take and "ack implosion". Reliable multicast has taken many forms: everyone must get everything, reliable for a specific subset of the receivers, reliable for any subset of the receivers, etc. A similar range of behavior has been proposed for flow control. In addition, protocols with a single sender appeared to allow simplifications that would not be possible in protocols with multiple senders. These sorts of proposals in a variety of combinations have manifested themselves in distinct protocol proposals. This has greatly clouded the nature of multicast transport and have made the problem look fairly complex.

The problem of "ack implosion" refers to potential performance bottleneck of acks and flow control credits being sent by all receivers to the sender of a multicast message. Everyone has been looking for a way to avoid the potential bottleneck at the sender from all of the receivers sending acks and credits back. Various schemes have been proposed none of them satisfactory. Once we recognize that transport mechanisms are all feedback mechanisms, it is apparent that there can be no solution. Some form of ack implosion is
inherent in the nature of the mechanism. While the schemes that had been proposed reduced the number of messages that must be returned to the sender, the amount of information must remain the same. The amount of information that must be fed back can be further decreased by different policies, such as defining reliable to mean that only a subset of the population has to receive the data, etc. We come to the conclusion that: For a reliable multicast transport protocol while data transfer is multicast, control is unicast. This basically means that we can define any form of multicast transport, i.e. whatever degree of reliability necessary, by specifying new policies for a traditional unicast transport protocol and using multicast addresses for data transfer.  

5.3. Transport Protocol to Cover Broad Operating Range

For some time, there has been considerable interest in new transport protocols primarily to support new networking services. OSI even found it necessary to define 5 different transport protocols which they called different classes. The operating characteristics of transport protocols are primarily determined by the requirements of applications. Therefore, it is important for a transport protocol to be able to accommodate the whole range of requirements of applications effectively and simply. For example, many have advocated that there was a requirement for new “high-speed” transport protocols to support high bandwidth networks. High speed networks never represent a requirement for high speed transport protocols. The bandwidth requirements for transport are determined by the needs of high speed applications.

More correctly, the requirement is for a transport protocol that can accommodate a wide range of speeds. With minor exceptions, the changes proposed to existing protocols to accommodate greater speed can be accomplished by changing policy. This can be a major advantage: a single protocol whose characteristics can be optimally matched to an application’s requirements by changing policy. In fact, if the application’s use changed rates, it would be possible to change policies to adjust the operation of the protocol without interrupting the connection.

5.4. Tailor Protocol Behavior to Traffic Characteristics and more Useful QoS

There has been very little progress in the last 20 years on incorporating quality of service into protocols. There has been much work on the kinds of QoS parameters that one would like to have, but there has been almost no work on how to incorporate these in any effective way into a real protocol design. These results give a clear indication as to how this might be accomplished: Changing values of QoS parameters must affect the policies of the protocol. Therefore, we can make the requirement that if a proposed QoS parameter cannot be related to changes in policy of a protocol in one of the lower layers, it is not really a QoS parameter. It now becomes an issue of developing the parameters and the policies that can be used to create the desired behavior in the protocol. This should be a straightforward experimental effort to carry out.

5.5. A Powerful Research Tool

As indicated by the previous point, an error control protocol developed along these lines becomes a powerful research tool into the behavior of error control protocols. One could

7. The realization that there is no way to avoid ack implosion is similar to the realization that one cannot build a perpetual motion machine. Once one knows what can’t be done, it is much easier to determine what can be done.

8. The exceptions are proposals, such as to modify the format of the header to make it faster to process, etc. These proposals have been contested as not providing much greater efficiency.

9. Some might say wishful thinking.
develop an experimental test implementation that would allow experiments with a wide variety of policies and mechanisms. Several of which have been alluded to in the course of this report. One of the biggest problems in protocol research up to now has been that one has been forced to come up with an entirely new protocol to test most proposals. This has made it difficult to isolate causes and effects and to do experiments where one was changing a "single variable." With a protocol developed along these lines, one could perform a wide range of experiments making well-defined changes in policy and be sure of not inadvertently introducing change elsewhere. One could conduct experiments in the trade-offs between acking and flow control, the effectiveness of ack vs nacking, etc. It would also be useful in experiments to better understand the interactions between various policies and its affects on protocol and system performance, and to establish the operating regions of particular combinations of policies.

6. Conclusions
This experiment started with the conjecture that separating mechanism from policy would create a much more effective approach to the design of error control protocols, such as those found in the data link and transport layers. That conjecture has been borne out by this work. Further, the methodology used to specify the mechanisms and policies also allowed us to uncover some heretofore unrecognized properties in the structure of protocols. These properties will have a profound affect on the implementation strategies adopted in the future. While it was the architecture work that indicated that the problem of multicast transport was not as difficult as we were making it, this work verified that conclusion by developing a set of policies that can be applied to a traditional transport protocol to produce a reliable multicast transport protocol. This in itself is a major contribution to the field and has major implications on the future development of multicast transport protocols.

This work doesn't solve all of the problems, but it does point in the direction of a much more methodical and scientific approach to understanding the behavior of protocols than has been taken up to now.

7. Future Work
This work has been just the first step. Even more so than most, this work opens the door for much future work:

- **Apply this Approach to TP4** – This approach does not require the design of a new protocol. It can be applied to an existing protocol. For example, one could, in essence, take TP4 apart, separate mechanism from policy, add an optional context field to the establishment mechanism and put it back together. If the context field was omitted, the protocol would adopt the "default" context, i.e. the policies defined for TP4 by the current specification. If the context field were present, the protocol would adopt the policies specified by the context, which might define multicast transport. This provides an effective means to introduce new capabilities into an existing protocol while maintaining backward compatibility. The result would be a TP4 that could support the same applications it does today, support multicast, support high speed and a wide range of new applications.

10. It is also interesting that had we concentrated on the multicast problem we would not have found this solution. This was one of those rare cases where only by starting with the more general problem did a simpler solution to the specific problem come to light.
• **Multicast** – Develop a set of policies aimed specifically at various forms of multicast within the TP4 structure, especially those forms found most commonly in Distributed Interactive Simulation.

• **Develop Policies for all of the above** – It would be very useful to develop a library of policies for the mechanisms that could be used with an error control protocol. The next step would be to characterize different combinations of policies that can be used together to achieve specific protocol behavior in support of particular applications or traffic conditions.

• **Develop an Experimental Program Based on this Tool** – As indicated above, one could develop a comprehensive research program that could take much of the mystery out of protocol design and operation.
Appendix A

Guidelines for the Specification of Services, Protocols, Mechanisms and Policies

Introduction

There are basically four kinds of LFMs: The Service of a Subsystem, protocol, mechanism, and policy. In a very real sense, there is really only one protocol: the generic protocol to which mechanisms and policies are added. For the lower layers, there is really only one service which varies in terms of the QoS it is able to provide.

Part I.
Layer Service Definitions

0. Introduction

1. Scope

2. References

3. Definitions

4. Compliance with other Specifications

5. Overview of the Service

5.1 Narrative Description of the Service

5.2 Basic Service and options

6. Detailed Specification of the Service

6.x Name of the Service Primitive

6.x.1 Description of the Function

6.x.2 Conditions for generating (this primitive)

   <this section should contain both the prose description and the formal description>

6.x.3 Action on receipt (of this primitive)

   <this section should contain both the prose description and the formal description>

6.x.4 Semantics of the parameters (of the primitive)

   <this section should contain a complete specification (or a pointer to a complete specification) of the parameters in the primitive>
Part II
Protocol Specifications

0. Introduction

1. Scope

2. References

3. Definitions

4. Compliance with other Standards

5. Overview of the Protocol
   5.1 Narrative Description of the Protocol
   5.2 Reference to the Service Definition
   5.3 Services Required by this Protocol
   5.4 Basic Protocol and Extensions

6. Detailed Specification of the Protocol
   6.1 Detailed specification of Common Functions
      (Sub sections giving a detailed specification of common functions following the
      style used below for the PDUs in 6.2).

   6.2 Detailed Specification of PDU Behavior
      6.2.i.1 Name of PDU
      6.2.i.2 Function
      6.2.i.3 Conditions for Generating (this PDU)
         6.2.i.3.1 Prose Description
         6.2.i.3.1 Formal Description
      6.2.i.4 Action on Receipt (of this PDU)
         6.2.i.4.1 Prose Description
         6.2.i.4.1 Formal Description

   6.3 Syntax of PDUs

   6.4 Mapping to Lower Service

7. State Tables

8. Conformance Requirements
Part III
Mechanism Specifications

1. Introduction

2. Facility Provided

3. Narrative Description of the Mechanism

4. Mechanism Identifier

5. LFM Mechanism Specification
   5.1. Other LFM$s Required
       5.2. Imports from other Mechanism Specifications:
           5.2.1. Local State Variables
                   Sending
                   Receiving
           5.2.2. Extended PDUs
           5.2.3. Other Syntactic Constructs
       5.3. Environmental Variables
       5.4. Additional PCI Required
           5.4.1. Syntax
           5.4.2. Semantics of the Fields
       5.5. Local State Variables
           Sending
           Receiving
       5.6. Additional Actions Associated with the Generic Protocol
           5.6.1. Data.submit
                   5.6.1.1. Informal Specification
                   5.6.1.2. Formal Specification
           5.6.2. Data PDU
                   5.6.2.1. Informal Specification
                   5.6.2.2. Formal Specification
       5.7. Actions Associated with Imported PDUs
5.8. New PDUs or Timers Associated with this Mechanism

5.8.1. New PDU Name
5.8.1.1. Description

5.8.1.2. When Generated
  5.8.1.2.1. Informal Specification

5.8.1.3. Action
  5.8.1.3.1. Informal Specification

5.8.1.4. Syntax

5.8.1.5. Semantics of the Fields

5.8.2. New Timer Name
5.8.2.1. Description

5.8.2.2. When Activated

5.8.2.3. Action upon Expiration
Part IV
Policy Specifications

1. Introduction
2. Policy Definition
3. Policy Identifier
4. Description of the Algorithm
5. Policy Specification
   5.1. Globals Used
       5.1.1. Local State
       5.1.2. PCI
   5.2. Policy Specification
       4.2.x Specification of Procedures
Appendix B

Generic Data Transfer
Service Definition
Covering Unicast & Multicast
Generic Data Transfer
Service Definition
Covering Unicast & Multicast

I. Introduction

The purpose of this document is to define the Generic Data Transfer Service that is provided among communicating peers to exchange information. The Generic Data Transfer Service definition is the complete representation of the service provider as viewed by its service users. The user of the service must incorporate the service definition into its operation to make proper use of the Generic Data Transfer Service. Hence the service definition constrains the behavior of the user of the Generic Data Transfer Service.

This service definition specifies the allocation and data transfer phases for a Generic Data Transfer Service. The service for the Enrollment Phase is defined separately. This service is a subset of all services whether for data transfer, layer management or applications. To create services specific to other areas, the generic data transfer phase may be further subdivided into other phases and additional service primitives may be added. Various forms of Data Transfer services are created by placing limits on the ranges of QoS supported. (For Data Transfer services, the variability is introduced in the mechanisms supported by various modules which define the behavior of protocol mechanisms and the policies associated with them. For Data Transfer Services, the behavior of these mechanisms is not generally visible above the service boundary.)

The Allocation phase consists of two important functions: 1) notifying the peer service user(s) that a peer has requested service and 2) establishing sufficient shared state among the peer protocol machines supporting the association to provide the requested QoS. A major aspect of this later facility is exchanging the context information among the peer protocol machines supporting this service. This context information specifies the policies to be enforced on the connection among the protocol machines to support the Generic Data Transfer Service for the service users.

The two important facilities of the Data Transfer Phase are: 1) the transparent transfer of data among the service users, and 2) the ability to change the context of the association.

The Generic Data Transfer Service definition also specifies in detail the service primitive actions and the allowed sequence of events of the service, the parameters associated with each primitive, and the interactions between the service user and the service provider while exchanging these service primitives at the service boundary.

II. References


III. Definitions

1. **Generic Data Transfer Service**: The fundamental data transfer service that is provided to service users at the boundary between the service provider and the service user.

2. **Service Provider**: An abstract representation of the protocol machine that provides Data Transfer Service to service users.

3. **Service User**: An abstract representation of the protocol machine or process that makes use of the Data Transfer Service.

4. **Service Primitive**: An abstraction of an interaction between a service user and service provider at the service boundary.

5. **OSI local view**: The shared behavior of the service provider and the service user in terms of their interactions at a service boundary.

6. **Symmetrical Service**: A service characterized by the behavior of all local views being the same and thus can be defined by the behavior of only one OSI local view.

7. **Asymmetrical Service**: A service characterized by the behavior of local views falling into multiple sets of local views with the same behavior.

8. **Submit(primitive)**: The service primitive issued by the service user for communicating with its service provider.

9. **Deliver(primitive)**: The service primitive issued by the service provider for communicating with its service user.

IV. Abbreviations
V. Compliance With Other Specifications

This document is in compliance with other specifications such as ISO 8072, ISO7498, DIS10731 and uses the method outlined in An Example of the Abracadabra service definition.

VI. Overview of the service

1. Narrative Description of the service

The Generic Data Transfer Service provides each communicating peer, the means to establish sufficient shared state for the exchange of data with a given QoS. At the time of allocation, each peer can request, and negotiate a certain level of QoS which is specified in the QoS parameters.

The Generic Data Transfer Service supports primitives for
- establishment of shared state such as Allocate.submit and Allocate.deliver
- termination of shared state such as Deallocate.submit and Deallocate.deliver
- data transfer such as Data.submit and Data.deliver

Communication for data transfer can be initiated by any peer within the scope of the service and within the constraints established by the enrollment phase. When a service user wishes to establish communication, it invokes the Allocate.submit primitive with the appropriate parameters to notify its service provider of its request for communication resources. The service provider upon receipt of this request, communicates this information to its peer to establish shared state through the Allocate.deliver primitive. The peer service user then informs its service provider of its acceptance by invoking the Allocate.submit primitive. The peer service provider notifies its peer protocol machine to complete the establishment of the shared state and to notify its service user that its peer is ready to accept data through the Allocate.deliver primitive. These interactions are illustrated in the following Fig a.
Communication establishment may be refused due to a number of conditions at the peer. The Deallocate.submit primitive from the service user, indicates that the service user is terminating the Allocation phase. The service user at the initiating end can also abandon the communication establishment by issuing the Deallocate.submit primitive. These interactions are illustrated in the following Figures b & c.
Once the allocation phase is complete, the data transfer can begin. Any communicating peer can initiate data transfer by invoking the Data.submit primitive, which is delivered as the Data.deliver primitive at the other end. Data transfer is illustrated in the following Fig d.

The established communication can be terminated by any communicating peer by issuing a Deallocate.submit primitive. Once a communication is terminated, any communicating peer may establish a new communication by issuing the Allocate.submit primitive.
1.2 Overview of interactions

The service user and the service provider interact with each other at the service boundary. These interactions form an abstract interface at the service boundary. This interface is termed as the OSI local view. The OSI local view is described in terms of the service primitives that are invoked by the service and service provider.

There are two kinds of service primitives.

1. Submit primitive issued by the service user to convey necessary information to its service provider.

2. Deliver primitive that is invoked by the service provider to communicate the necessary information to its service user.

Generic Data Transfer Service defines the above two types of service primitives for communication establishment (Allocate.submit, Allocate.deliver), data transfer (Data.submit, Data.deliver), changing policies (Context.submit, Context.deliver, Context-Rsp.submit, Context-Rsp.deliver) and communication termination (Deallocate.submit, Deallocate.deliver).
VII. Description of the states of the Generic Data Transfer Service

The Generic Service defines the following states of the OSI local view:

1. **Idle**: OSI local view is said to be in idle state, when there is no communication established OR there is no attempt to establish a communication.

2. **Association pending**: OSI local view enters this state either when a communication has been requested by the service user and is waiting for the response from its peer OR the service user is informed by its service provider about a request for communication from its peer and is waiting for the response from its service user.

3. **Data-transfer-ready**: OSI local view enters this state, once the communication establishment phase is complete.

VIII. State Transition diagram for the possible sequence of primitives

![State Transition Diagram](image)

Fig I. Sequence of Service primitives at OSI local view
**IX. Description Of the Service**

The following table summarizes the Generic Data Transfer Service Primitives

<table>
<thead>
<tr>
<th>Phase</th>
<th>Service</th>
<th>Primitives</th>
<th>Parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td>Allocation</td>
<td>Communication establishment</td>
<td>Allocate.submit</td>
<td>Source-address, Destination-address, Context-identifier, QoS, User-data, Context-id/QoS</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Allocate.deliver</td>
<td>Source-address, Destination-address, Context-identifier, QoS, User-data, Context-id</td>
</tr>
<tr>
<td>Allocation</td>
<td>Communication termination</td>
<td>Deallocate.submit</td>
<td>User-data</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Deallocate.deliver</td>
<td>User-data</td>
</tr>
<tr>
<td>Change in Policy</td>
<td>Change in Policy</td>
<td>Context.submit</td>
<td>QoS, Context-identifier</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Context.deliver</td>
<td>Context-identifier</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Context-Rsp.submit</td>
<td>Context-identifier, Result</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Context-Rsp.deliver</td>
<td>Context-identifier, Result</td>
</tr>
<tr>
<td>Data Transfer</td>
<td>Data Transfer</td>
<td>Data.submit</td>
<td>User-data</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Data.deliver</td>
<td>User-data</td>
</tr>
</tbody>
</table>

**Destination-address:** The destination-address in the Allocate.submit primitive (request for communication) might specify a set of addresses and the Allocate.deliver primitive (response for communication request) would have a destination-address that chose one of the sets which will be used for this communication.

**QoS:** The Allocate.submit primitive (request for communication) might specify certain QoS parameters to be maintained during the communication. The Allocate.deliver primitive (response for communication request) would have the QoS parameters that are agreed by the communicating peers from which communication has been requested.
1. Parameters associated with each service primitive

1.1 Source-address: This parameter indicates the address of the communicating peer issuing the allocation primitive.

1.2 Destination-address: This parameter indicates the address of the communicating peers which are to be affected by the issue of allocate primitive.

1.3 Context-identifier: This parameter is used to indicate the policy being enforced during communication establishment.

1.4 User-data: This parameter indicates the user data messages to be transferred over the communication established between the two service users.

1.5 Quality of service(QoS): The quality of service refers to certain characteristics that are associated with the communication management of the Generic Data Transfer Service. QoS is offered in terms of QoS parameters. The QoS is usually negotiated between the communicating entities for each communication, using the communication establishment service primitives.

1.6 AGI: The AGI parameters that will be associated with allocate primitives are those that are dynamic and are selected at the time of communication allocation. AGI is all enrollment phase parameters. Parameters classified as AGI in Allocate or Data transfer are really multicast QoS.

2. Description of the service primitives:

2.1. Allocate.submit (Source-address, Destination-address, Context-identifier, QoS, User-data)

2.1.1 When invoked: When the OSI local view is in the Idle state, this primitive is invoked by the service user to inform its service provider of its request to establish communication with its peer (whose address is specified in the Destination-address parameter). The service user must also specify the QoS parameter that it needs for the communication.

When the OSI local view is in the Association pending state, this primitive is invoked by the service user to inform the service provider of its intent to accept the request for communication (which is delivered to the provider through the Source-address parameter). The service user must also agree to the QoS specified by the peer.

2.1.2 Action: If the OSI local view is in the Idle state, then the service provider must inform the peer service provider (whose address is specified in the Destination-address parameter) of the request for communication. OSI local view now enters the Association pending state.

When the OSI local view is in the Association pending state, the service provider upon receipt of this primitive from its service user must inform the provider (whose address is indicated in the source-address parameter) about its acceptance of the request for communication. Now the OSI local view enters the Data-transfer-ready-state. The service provider must also inform its peer about the QoS that it will provide once communication is established.

2.2. Allocate.deliver(Source-address, Destination-address, Context-identifier, QoS, User-data)
2.2.1 When invoked: When the OSI local view is in the Idle state, this primitive is invoked by the service provider to inform its service user about the request for a communication that it has received from the peer (whose address is specified in the Source-address parameter). OSI local view now enters the Association pending state. The service provider must also notify its service user about the QoS required by the peer.

When the OSI local view is in the Association pending state, the service provider issues this primitive to inform its service user as to its acceptance of the communication request by the peer (whose address is specified in the Destination-address parameter). OSI local view now enters Data-transfer-ready state.

2.2.2 Action: When the OSI local view is in the idle state and receives the Allocate.deliver, the service user must decide whether it intends to accept the peer's (whose address is specified in the Destination-address parameter) request for communication. If the service user wishes to accept the request, then it must invoke the Allocate.submit primitive with the necessary parameters. The service user must issue the Deallocate.submit primitive, if it wishes to reject the request for communication.

When the OSI local view is in the Association pending state, the service user upon the receipt of this primitive may invoke the Data.submit primitive with User-data parameter. OSI local view now enters the Data-transfer-ready state.

2.3. Deallocate.submit(User-data)

2.3.1 When Invoked: The service user issues this primitive to terminate a communication OR to reject a request for communication

2.3.2 Action: The service provider upon receipt of this primitive must invoke the necessary mechanism to terminate the communication. The service user does not receive any further data. OSI local view now enters the Idle state.

2.4. Deallocate.deliver(User-data)

2.4.1 When Invoked: The service provider issues this primitive to notify its service user about the termination of the data transfer phase OR to refuse the request for communication. OSI local view now enters the Idle state.

2.4.2 Action: The service user must not issue any other service primitives other than Allocate.submit. OSI local view now enters the Idle state.

2.5. Context.submit(Context-identifier)

2.5.1 When Invoked: The service user invokes this primitive to request its service provider for the change in policy (which is specified in the Context-identifier parameter).

2.5.2 Action: Upon receipt of this primitive, the service provider must inform its peer about the request for a change in policy.

2.6. Context.deliver(Context-identifier)

2.6.1 When Invoked: This primitive is issued by the service provider to inform its service user about the request for change in policy (which is specified through Context-identifier parameter) that it has received from its peer.
2.6.2 **Action:** If the service user wishes to accept the request, it must invoke the Context-Rsp.submit primitive.

2.7. **Context-Rsp.submit(Context-identifier, Result)**

2.7.1 **When Invoked:** This primitive is invoked by the service user to inform its service provider of its acceptance of the request for change in policy.

2.7.2 **Action:** Upon receipt of this primitive, the service provider must inform its peer about the acceptance of its request for change in policy by its service user.

2.8. **Context-Rsp.deliver(Context-identifier, Result)**

2.8.1 **When Invoked:** The service provider invokes this primitive to inform its service user about the acceptance of its request for change in policy. Service provider must also invoke necessary mechanisms to change policy.

2.8.2 **Action:** This must be decided at the local level.

2.9. **Data.submit(User-data)**

2.9.1 **When Invoked:** The service user issues this primitive to request its service provider for transferring User-data.

2.9.2 **Action:** When the service provider receives this primitive, OSI local view must be in the Data-transfer-ready state. Otherwise it is an error. Upon receipt of this primitive, the service provider must initiate the necessary mechanisms to transfer User-data in manner consistent with the requested QoS.

2.10. **Data.deliver(User-data)**

2.10.1 **When Invoked:** The service provider invokes this primitive to deliver the User-data that it has received for its service user.

2.10.2 **Action:** The requirements that the service user has to meet upon the receipt of this primitive are a local matter.
Appendix C

Generic Data Transfer Protocol Specification
I. Introduction

The purpose of this document is to specify the Generic Data Transfer protocol for the transfer of data among communicating peers. This protocol is a companion to the Generic Data Transfer Service definition. The protocol specifies only the data transfer phase. The allocation and deallocation phases are specified separately in the Allocation Mechanism.

The two important facilities of the Data transfer phase are: 1) the transparent transfer of data among the communicating peers and 2) the ability to change the context of the association.

This protocol specification also presents the details of the DATA PDU which is used to exchange the user data among the communicating peers. The common functions that are used for sequencing of DATA PDUs, Flow Control and PDU based error control are presented in separate mechanism.

II. References


[DIS 10731] Information technology - Open systems interconnection - Basic Reference model


III. Definitions
This protocol specification makes use of the following terms defined in the Generic Data Transfer Service Definition:

1. Service primitive
2. Submit(primitive)
3. Deliver(primitive)
4. Generic Data Transfer Service
5. Protocol data unit(PDU): a unit of data specified in a protocol and consisting of certain protocol control information and possibly some user data messages.

IV. Compliance with other standards

This document is in compliance with other specifications such as ISO 8072, ISO7498, DIS 10731 and uses the method outlined in An Example of the Abracadabra Protocol Specification.

V. Overview Of the Protocol:

1. Narrative Description Of the Protocol:

The Generic Data Transfer protocol specifies the actions of the protocol entities during the transfer of user data among the communicating peers. The protocol supports Data.submit and Data.deliver primitives for the transfer of user data.

Once the allocation phase has been completed among the communicating peers, data transfer can begin. Data transfer is initiated by invoking the Data.submit primitive with DATA PDU. Sending Data PDUs will depend on various loosely coupled mechanisms, such as flow control, that may be in effect. Tightly coupled mechanisms such as sequencing and data corruption are applied to the PDU before sending, if they are in effect. The Data PDU is delivered at the receiving end through Data.deliver primitive. Upon receipt of the DATA PDU, the receiver must execute any tightly coupled mechanisms included in the protocol in the specified order. Data transfer phase is illustrated in the following fig.
2. References to the Service Definition

The Generic Data transfer protocol refers to the same Data transfer primitives such as Data.submit and Data.deliver Which are specified in the Generic Data Transfer service Definition The Data transfer phase of this protocol is also closely related to the Data transfer phase of the service definition.

VI. Specification of Common Functions

1. Sequencing of PDUs and PDU based error control

The functions that provide sequencing of PDUs and PDU based error control during Data transfer phase are specified by separate mechanism specifications such as unicast sequencing mechanism and error control mechanism.

2. Detailed Specification of PDU Behavior

2.1. Name of the PDU: DATA

2.2. Function

This PDU is used to transfer user data among the protocol entities.

2.3. When Sent

The protocol entity sends this PDU whenever it is in the data transfer state, and the conditions of any loosely coupled mechanisms in effect allow the sending of the DATA PDU.

2.4. Action on Receipt (of this PDU)

If the protocol entity is in the data transfer and it receives a DATA PDU, the receiver must execute any tightly coupled mechanisms included in the protocol in the specified order.

If the protocol entity is not in the data transfer state, it signals an error.
Appendix D

Unicast Sequencing
Mechanism Specification
and Example Policies
1. **Introduction**

This document specifies the additional mechanisms required to provide sequencing for a unicast error control protocol, such as those found at the Transport Layer. Since some relaying protocols may route PDUs along multiple paths with different delay characteristics, the PDUs may not arrive in the same order in which they were sent. This mechanism provides the facility necessary to re-order them.

2. **Facility Provided**

This mechanism allows the protocol to provide various degrees of local ordering by allowing the receiver to reconstruct the order in which the PDUs were sent by the source. The Sequencing Policy defines what action is taken when a PDU is received out of order.

3. **Narrative Description of the Mechanism**

The Unicast Sequencing Mechanism operates by assigning a sequence number to each PDU sent. The Sequence Numbers are monotonically increasing, increasing by one for each PDU sent. The Receiver will immediately deliver any PDU arriving in order. If PDUs arrive out of order, the Sequencing Policy is invoked to take the appropriate action. A queue, the OutOfOrderQ, is provided for ordering out of order PDUs until the Sequencing Policy determines they should be delivered. The Sequencing Policy is not required to utilize this queue.

4. **Mechanism Identifier**

Unicast Sequencing: <Identifier to be assigned>

5. **Mechanism Specification**

5.1. **Other Mechanisms Required**

This Mechanism assumes the Generic Allocation Protocol.

5.2. **Imports from other Mechanism Specifications**

This Mechanism does not import other variables from other Mechanisms.

5.2.1. **Local State Variables**

Not Required

5.2.2. **Extended PDUs**

AllocateRQ PDU
AllocateRsp PDU
AllocateCnf PDU

---

1. It would be a minor adjustment of this mechanism to change the units of the sequence numbers to be octets, rather than PDUs. However, using units of octets is only recommended for relative low speed protocols, since the finer granularity impairs performance by requiring larger PCI and more complex error and flow control, if these mechanisms are present.
DeallocateRQ PDU
DeallocateRsp PDU

If present:
Set Context RQ PDU
Set Context Rsp PDU

5.2.3. Other Syntactic Constructs
Not Required.

5.3. Environmental Variables
SeqNbrUnits  – Defines the units by which the sequence number is to be incremented. The two common choices are in terms of PDUs or octets.

SeqNbrLength  – Defines the Length in bits of the SequenceNumber field in the PCI. All calculations on sequence numbers are performed modulo this quantity.

5.4. Additional PCI Required
5.4.1. Syntax
Sequence Number: Integer(SeqNbrLength)

5.4.2. Semantics of the Fields
Sequence Number  – This field contains the sequence number of this PDU. The sequence number indicates the position in the data stream of this PDU. PDUs on this connection with lower sequence numbers were sent before this one and PDUs with higher sequence numbers were sent after it.

This PCI field is included in the Data PDUs of the Generic Protocol. If the Context Negotiation Mechanism is used, its PDUs will include this PCI field.

5.5. Local State Variables
Sending
NextSeqtoSend  – This state variable will contain the Sequence Number of the Next Sequence to be assigned to a PDU sent on this connection.

Receiving
NextSeq  – This state variable will contain the Sequence Number of the next in order PDU expected to be received on this connection. This is traditionally referred to as the Receiver’s Left Window Edge. The receiver does not expect to receive any PDUs with SequenceNumbers less NextSeq. (Such PDUs would be duplicates.)

OutOfOrderQ  – A queue into which messages received out of order are inserted in Sequence Number order. The decision to insert PDUs into this queue is determined by Sequeencing Policy.

5.6. Additional Actions Associated with the Generic Protocol
5.6.1. Data.submit
5.6.1.1. Informal Specification
Set SequenceNumber of Set Context PDU to NextSeqtoSend.
Increment NextSeqtoSend by 1.
5.6.1.2. Formal Specification

5.6.2. Data PDU

5.6.2.1. Informal Specification

If SequenceNumber = NextSeq Then
   Perform any operations associated with this PDU (For Data, strip off header; for Set Context or Context Rsp, put new policies into effect)
   This PDU can be delivered; subsequent to operations by other mechanisms
   NextSeq:=NextSeq+1;
   While SequenceNumber of the PDU on the OutofOrderQ = NextSeq or Until Q is empty Do
      Take the next message off the OutofOrderQ
      Perform any operations associated with this PDU (For Data, strip off header; for Set Context or Context Rsp, put new policies into effect)
      This PDU can be delivered; subsequent to operations by other mechanisms
      NextSeq:=NextSeq+1
   Od;
Else
   Invoke Sequencing Policy.
Fi

5.6.2.2. Formal Specification

5.7. Actions Associated with Imported PDUs

5.7.1. Set Context.submit

5.7.1.1. Informal Specification

Set SequenceNumber of Set Context PDU to NextSeqtoSend.
Increment NextSeqtoSend by 1.

5.7.1.2. Formal Specification

5.7.2. Set Context Response.submit

5.7.2.1. Informal Specification

Set SequenceNumber of Set Context Rsp PDU to NextSeqtoSend.
Increment NextSeqtoSend by 1.

5.7.2.2. Formal Specification

5.7.3. Set Context RQ PDU

5.7.3.1. Informal Specification

If SequenceNumber = NextSeq Then
   This PDU can be delivered; subsequent to operations by other mechanisms
   NextSeq:=NextSeq+1;
   While SequenceNumber of the PDU on the OutofOrderQ = NextSeq or Until Q is empty Do
      Take the next message off the OutofOrderQ
      Perform any operations associated with this PDU (For Data, strip off header; for Set Context or Context Rsp, put new policies into effect)
      This PDU can be delivered; subsequent to operations by other mechanisms
      NextSeq:=NextSeq+1
   Od;

D-3
Else
   Invoke Sequencing Policy.
Fi

5.7.3.2. Formal Specification

5.7.4. Set Context Rsp PDU

5.7.4.1. Informal Specification
If SequenceNumber = NextSeq Then
   Perform any operations associated with this PDU (For Data, strip off header; for Set Context or Context Rsp, put new policies into effect)
   This PDU can be delivered; subsequent to operations by other mechanisms
   NextSeq:=NextSeq+1;
   While SequenceNumber of the PDU
      on the OutofOrderQ = NextSeq or Until Q is empty Do
         Take the next message off the OutofOrderQ
         Perform any operations associated with this PDU (For Data, strip off header; for Set Context or Context Rsp, put new policies into effect)
         This PDU can be delivered; subsequent to operations by other mechanisms
         NextSeq:=NextSeq+1
         Od;
   Else
      Invoke Sequencing Policy.
   Fi

5.7.4.2. Formal Specification

5.8. New PDUs or Timers Associated with this Mechanism

No New PDUs defined for this Mechanism.
Null Sequencing
Policy Specifications

1. Introduction
This policy is a policy to be used with the Sequencing Mechanism in error control protocols built upon the Generic Protocol. This Mechanism and Policy extend the functionality of both the Allocation and Data Transfer Phases.

2. Policy Definition
This policy simply discards the PDU. The Policy is used for those situations that error control will be used to fill the gaps or QoS does not require all PDUs to be delivered.

3. Policy Identifier
TBD

4. Description of the Algorithm
Discard the PDU.

5. Policy Specification

5.1. Globals Used
None

5.2. Policy Specification
5.2.1. Specification of Procedures
Discard the PDU.
The Common Sequencing
Policy Specifications

1. Introduction
This policy is a policy to be used with the Sequencing Mechanism in error control protocols built upon the Generic Protocol. This Mechanism and Policy extend the functionality of both the Allocation and Data Transfer Phases.

2. Policy Definition
This policy defines what is probably the most commonly used Sequencing Policy. The PDUs are placed on the OutofOrderQ up to some maximum after which PDUs are discarded.

3. Policy Identifier
TBD

4. Description of the Algorithm
The Policy is very straightforward. PDUs are placed on the OutofOrderQ in increasing SequenceNumber order. However the Q has a finite capacity. The Q is full and PDUs arrive they are discarded. Note: If a Flow Control Mechanism and Policy are also in use, it should be aware of the number of PDUs in the OutofOrderQ (as well as PDUs queued for the layer above) and lower the allocation extended to the sender when appropriate.

5. Policy Specification
5.1. Globals Used
5.1.1. Local State
OutofOrderQ – (see the Sequencing Mechanism Specification.)
MaxPDUQd - This variable defines the maximum number of PDUs that can be placed on the OutofOrderQ. This variable reflects the amount of buffer space available for queuing PDUs. This may reflect a pool of buffers and the fraction of the pool that may be available to this connection.

5.1.2. PCI Used
SequenceNumber – (see the Sequencing Mechanism Specification.)

5.2. Policy Specification
5.2.1. Specification of Procedures
If Number of PDUs on the OutofOrderQ < MaxPDUQd Then
Place the PDU on the OutofOrderQ in increasing SequenceNumber order
else
Discard the PDU
Fi
1. Introduction
This policy is a policy to be used with the Sequencing Mechanism in error control protocols built upon the Generic Protocol. This Mechanism and Policy extend the functionality of both the Allocation and Data Transfer Phases.

2. Policy Definition
This Policy sequences PDUs as they arrive, but allows gaps in the data stream to occur as long as they are not too large. The Policy is used for those situations where QoS does not require all PDUs to be delivered.

3. Policy Identifier
TBD

4. Description of the Algorithm
This policy discards a PDU that has a sequence number less than the last seen. Any PDU delivered out of order greater than the last sequence number seen and less than some maximum gap size is simply delivered. If the gap is too large, the QoS has been violated and an error is generated.

5. Policy Specification
5.1. Globals Used
5.1.1. Local State
OutofOrderQ – (see the Sequencing Mechanism Specification.)
MaxGap – The maximum size of a gap in the PDUs received that can be tolerated expressed in terms of sequence numbers.

5.1.2. PCI Used
SequenceNumber – (see the Sequencing Mechanism Specification.)

5.2. Policy Specification
5.2.1. Specification of Procedures
If SequenceNumber < SequenceNumber+MaxGap Then
   Deliver to the User any PDUs on the OutofOrderQ with a SequenceNumber less than this one (canceling all timers up to this SequenceNumber) followed by this PDU.
Else
   Put the PDU on the OutofOrderQ in increasing SequenceNumber order
   Set Timer to return SequenceNumber in a time approximating MaxGap
Fi

When Timer expires, indicate an error. The Quality of Service has dropped below acceptable levels.

Partial Ordering
Policy Specification
Appendix E
Data Corruption Detection
Mechanism Specification
Data Corruption Detection
Mechanism Specification

1. Introduction
This document defines the Data Corruption Detection mechanism. This Mechanism allows
the receiving Protocol Machine to detect various forms of errors that may have occurred dur-
ing transmission.

2. Facility Provided
This Mechanism is used to detect PDUs that may have been corrupted, i.e. bit errors, during
transmission. By selecting different policies, it is possible to adjust the kinds of errors that
are detected and the action taken when an error is detected. The mechanism also allows the
PCI and User-data to protected independently.

3. Narrative Description of the Mechanism
The Mechanism includes two error code (or checksum) fields in each PDU of the allocation
and data transfer phases. The PDU error code field applies to the whole PDU. The PCI
Error Code applies only to the PCI of the PDU. When a corrupted PDU is detected, the ac-
tion taken is determined by policy. The mechanism may use one or both and have different
error codes for both.

4. Mechanism Identifier
Data Corruption Detection: <identifier to be assigned>

5. Mechanism Specification
5.1. Other Mechanisms Required
None

5.2. Imports from other Mechanism Specifications:
5.2.1. Local State Variables
Sending
None

Receiving
None

5.2.2. Extended PDUs
No additional PDUs are imported for this Mechanism.

5.2.3. Other Syntactic Constructs

5.3. Environmental Variables
None
5.4. Additional PCI Required

5.4.1. Syntax

PDUErrorCode(ErrorCode1Length
PCIErrorCode(ErrorCode2Length) Optional

5.4.2. Semantics of the Fields

PDUErrorCode – This field contains the error code computed as a function of the bytes in the PDU.

PCIErrorCode – This field contains the error code computed as a function of the bytes in the PCI. This optional field is provided for those cases where it is desired to have separate error codes on the PCI and User-data.

5.5. Local State Variables

Sending
None

Receiving
None

5.6. Additional Actions Associated with the Generic Protocol

5.6.1. Data.submit

5.6.1.1. Informal Specification
Invoke Sending Data Corruption Policy

5.6.1.2. Formal Specification

5.6.2. Data PDU

5.6.2.1. Informal Specification
Invoke Receiving Data Corruption Policy
If Error Detected Then
   Invoke Receiving Corrupted PDU Policy
Fi

5.6.2.2. Formal Specification

5.7. Actions Associated with Imported PDUs

For all PDUs imported to support mechanisms in the allocation and data transfer phases, actions are taken for each submit and PDU received that are identical to the actions for the Data.submit and Data PDU described above.

5.8. New PDUs or Timers Associated with this Mechanism

None

5.8.1. New PDU Name

5.8.1.1. Description
N/A

5.8.1.2. When Generated

5.8.1.2.1. Informal Specification
N/A
5.8.1.3. Action
   5.8.1.3.1. Informal Specification
       N/A

5.8.1.4. Syntax
       N/A

5.8.1.5. Semantics of the Fields
       N/A

5.8.2. New Timer Name
   5.8.2.1. Description
       N/A

   5.8.2.2. When Activated
       N/A

   5.8.2.3. Action upon Expiration
       N/A
Appendix F
Lost and Duplicate Detection
Mechanism Specification
and Example Policies
Lost and Duplicate Detection Mechanism Specification

1. Introduction
This document defines Lost and Duplicate Detection mechanisms to provide PDU based error control, i.e. lost and duplicate detection. This Mechanism does not detect corrupted PDUs. That function is provided by a separate mechanism. Since some relaying protocols may be subject to congestion, PDUs may be lost or duplicated by retransmissions. This Mechanism detects these lost and duplicate messages. The action taken once the errors are detected is a matter of policy.

2. Facility Provided
The Lost and Duplicate Detection Mechanism requires the Sequencing Mechanism to be in place. Based on the SequenceNumbers seen by the receiver, an AckPDU is sent to the sender. The AckPDU supports what has been traditionally called positive and negative acknowledgements. The term acknowledge is slightly misleading. The semantics of a positive acknowledgement is considered to be: "The receiver will not request retransmission of any PDUs with the Sequence Numbers indicated by the Ack." The semantics of the negative acknowledgement is to request the retransmission of the PDUs with sequence numbers or ranges of sequence numbers contained in the AckPDU. There are six possible policies that might be associated with this mechanism:

1) Retransmission Timer Policy - This policy is executed by the sender to estimate the duration of the retransmission timer. This policy will be based on an estimate of round-trip time and the Ack or Ack List policy in use.

2) Receiving Error Control Policy - This policy is executed by the receiver to determine when to positively or negatively ack PDUs. The Policy is executed upon receipt of any PDU protected by this Error Control mechanism.

3) Sending Ack Policy - This policy is executed by the Sender and provides the Sender with some discretion on when PDUs may be deleted from the ReTransmissionQ. This is useful for multicast and similar situations where one might want to delay discarding PDUs from the retransmission queue.

4) Sending Ack List Policy - This policy is executed by the Sender and provides the Sender with some discretion on when PDUs may be deleted from the ReTransmissionQ. This policy is used in conjunction with the selective acknowledgement aspects of the mechanism. This is useful for multicast and similar situations where one might want to delay discarding PDUs from the retransmission queue.

5) Sending NAck Policy - This policy is executed by the Sender and provides the Sender with some discretion on when PDUs may be retransmitted from the ReTransmissionQ.
6) Sending NAck List Policy - This policy is executed by the Sender and provides the Sender with some discretion on when PDUs may be retransmitted from the ReTransmissionQ. This policy is used in conjunction with the selective retransmission aspects of the mechanism.

3. Narrative Description of the Mechanism

There are two modes in which the Lost and Duplicate Detection Mechanism can be used: Positive Ack only, or Positive/Negative Ack. This mechanism applies to all PDUs generated in the Data Transfer Phase, i.e. Data and Set Context PDUs. The sender of PDUs will maintain a retransmission queue, i.e. each sender maintains such a queue. When a PDU is sent, a copy is placed on this RetransmissionQ and a timer is started. The timer has a value $T_R$, and is generally a function of round-trip time. (Calculation of the timer value is calculated by the Retransmission Timer Policy.) If the timer expires before an AckPDU is received, all PDUs on the RetransmissionQ are re-sent.

When a PDU is received, if the SequenceNumber of the PDU is less than the lowest sequence number on the retransmission queue or if there is a PDU on the OutOfOrderQ with this SequenceNumber, then this PDU is a duplicate and it is discarded. The Receiving Error Control Policy is then invoked to determine the appropriate action based on the SequenceNumber of the PDU. Ultimately, the Receiving Error Control Policy will generate an AckPDU to positively or negatively acknowledge the PDUs it has received. The AckPDU allows all PDUs before or after a particular Sequence Number to be positively or negatively acknowledged, or for ranges of sequence numbers to be selective acknowledged. The Ack/Nack flag indicates whether the single sequence number is to be considered an Ack or a Nack, while the Ack/Nack List flag indicates whether the Sequence Number List indicates Ack or Nack. The setting of the two flags is completely independent.

If the Ack/Nack flag is true then, the sender will delete all PDUs from the ReTransmissionQ with SequenceNumbers less than or equal to the contents of the Ack/Nack field and discontinue any Retransmission Timers associated with these PDUs. Otherwise (the Ack/Nack field is interpreted as a Nack), the sender will retransmit all PDUs on the RetransmissionQ with SequenceNumbers greater than or equal to the contents of the Ack/Nack field.

If the Ack/Nack List Length is non zero, then there are selective acks or nacks depending on the value of the Ack/Nack List flag. If the Ack/Nack List flag is true then, the PDUs designated by the ranges of SequenceNumbers indicated by the Ack/Nack List are deleted from the RetransmissionQ and any timers associated with these PDUs are discontinued. Otherwise (the Ack/Nack List is interpreted as a Nack), the PDUs from those ranges are retransmitted. Note that since one can never know whether a retransmitted PDU arrives, a Negative Ack can never cause PDUs to be removed from the RetransmissionQ.

4. Mechanism Identifier

Lost and Duplicate Detection: <Identifier to be assigned>

5. Mechanism Specification

5.1. Other Mechanisms Required

In addition to the basic mechanisms of the Generic Protocol, the Lost and Duplicate Detection requires the use of the Generic Allocation Mechanism and the Sequencing Mechanism.
5.2. Imports from other Mechanism Specifications

Imports from the Sequencing mechanism the following:

5.2.1. Local State Variables

Sending
NextSeqToSend – This state variable will contain the Sequence Number of the Next Sequence to be assigned to a PDU sent on this connection. This is traditionally referred to as the Sender’s Right Window Edge.

Receiving
NextSeq – This state variable will contain the Sequence Number of the next in order PDU expected to be received on this connection. This is traditionally referred to as the Receiver’s Left Window Edge. The receiver does not expect to receive any PDUs with SequenceNumbers less NextSeq. (Such PDUs would be duplicates.)

OutofOrderQ – A queue into which messages received out of order are inserted in Sequence Number order. The decision to insert PDUs into this queue is determined by Sequencing Policy.

5.2.2. Extended PDUs

AllocateRQ PDU
AllocateRsp PDU
AllocateCnf PDU
DeallocationQ PDU
DeallocationRsp PDU

If present:
Set Context RQ PDU
Set Context Rsp PDU

5.2.3. Other Syntactic Constructs

Not required.

5.3. Environmental Variables

The Units and Length of the SequenceNumber are used from the Sequencing Mechanism.

5.4. Additional PCI Required

5.4.1. Syntax

No additional PCI in the PDU of the Generic Protocol or in the PDUs of any imported PDUs are required by this mechanism.

5.4.2. Semantics of the Fields

N/A

5.5. Local State Variables

Sending
ReTransmissionQ – A queue on which DataPDUs or Set Context PDUs which have not been positively acknowledged are placed in SequenceNumber order. All DataPDUs and Set Context PDUs sent are placed on this queue. A Positive Ack will remove them.

NextAck – This state variable contains the Sequence Number of the lowest sequence number expected to be Acked. This should be the Sequence Number of the first PDU on...
the RetransmissionQ. (This is traditionally called the Sender’s Left Window Edge.) This variable is primarily provided as convenience, since one could always look at the first PDU on the queue and gain the same information.

\( T_R \) – The time interval for which the sender waits for a positive acknowledgement before retransmitting. This value may be set by network management, or by the Sending Error Control Policy (generally the Sending Error Control Policy would be based on one of the round trip estimation algorithms). Note: The Sending Control Policy must be coordinated with the Receiving Error Control Policy to ensure that an appropriate retransmission time is chosen.

ReXmitMax – The maximum number of retransmissions of PDUs without a positive acknowledgement that will be tried before declaring an error.

Receiving
None identified at this time.

5.6. Additional Actions Associated with the Generic Protocol

5.6.1. Data.submit

5.6.1.1. Informal Specification
A copy of each DataPDU generated as a result of a Data.submit is placed on the RetransmissionQ in FIFO order and a Retransmission timer associated with this PDU is started with the value \( T_R \).

The DataPDU is ready to send pending any operation by subsequent mechanisms and their policies.

5.6.1.2. Formal Specification

5.6.2. Data PDU

5.6.2.1. Informal Specification
If the SequenceNumber of the PDU \( \leq \text{NextSeq} \) or there is a PDU on the OutofOrderQ with this SequenceNumber Then Discard this PDU as a duplicate

Fi
Invoke the Receiver Error Control Policy.

5.6.2.2. Formal Specification

5.7. Actions Associated with Imported Service Primitives and PDUs

5.7.1. Set Context.submit

5.7.1.1. Informal Specification
A copy of each SetContextRQPDU generated as a result of a Set Context.submit is placed on the RetransmissionQ in FIFO order and a Retransmission timer associated with this PDU is started with the value \( T_R \).

The SetContextRQPDU is ready to send pending any operation by subsequent mechanisms and their policies.

5.7.1.2. Formal Specification

5.7.2. Set Context Response.submit
5.7.2.1. Informal Specification
A copy of each SetContextRspPDU generated as a result of a Set Context Response.submit is placed on the RetransmissionQ in FIFO order and a Retransmission timer associated with this PDU is started with the value $T_R$.

The SetContextRspPDU is ready to send pending any operation by subsequent mechanisms and their policies.

5.7.2.2. Formal Specification

5.7.3. Set Context RQ PDU
5.7.3.1. Informal Specification
If the SequenceNumber of the PDU ≤ NextSeq or there is a PDU on the OutofOrderQ with this SequenceNumber Then Discard this PDU as a duplicate
Fi
Invoke the Receiver Error Control Policy.

5.7.3.2. Formal Specification

5.7.4. Set Context Rsp PDU
5.7.4.1. Informal Specification
If the SequenceNumber of the PDU ≤ NextSeq or there is a PDU on the OutofOrderQ with this SequenceNumber Then Discard this PDU as a duplicate
Fi
Invoke the Receiver Error Control Policy.

5.7.4.2. Formal Specification

5.8. New PDUs or Timers Associated with this Mechanism

5.8.1. AckPDU

5.8.1.1. Description
This PDU is sent by the receiver of DataPDUs or Set Context PDUs to either request the sender to retransmit the PDUs with given sequence numbers or to indicate to the sender that no retransmissions of DataPDUs with the given sequence numbers will be required.

5.8.1.2. When Generated

5.8.1.2.1. Informal Specification
The AckPDU is generated when the Receiving protocol machine is in the Data Transfer Phase and by conditions determined by the Receiving Error Control Policy in force at the time.

5.8.1.3. Action

5.8.1.3.1. Informal Specification
When the Protocol Machine receives an AckPDU, it determines whether it is well-formed and if not the PDU is discarded; otherwise:
Invoke Retransmission Timer Policy.
If Ack/Nack flag is true Then
  If Sending Ack Policy Then
    For all PDUs on the RetransmissionQ with SequenceNumbers less than
    or equal to the value of the Ack/Nack field, delete from the queue and
    discontinue any Retransmission Timer associated with them.
  Else
    If Sending Nack Policy Then
      Retransmit all PDUs from value of the Max(Ack/Nack field, NextAck-1)
      to NextSeqtoSend-1
    Else
      If the Ack/Nack List Length ≠ 0 then
        If Ack/Nack List flag is True
          Do
            If Sending Ack List Policy Then
              For each pair of Starting and Ending SequenceNumbers,
              Delete this range of PDUs with these SequenceNumbers from the
              RetransmissionQ and discontinue any Retransmission Timer associated
              with them.
            Else
              Do
                If Sending Nack List Policy Then
                  For each pair of Starting and Ending SequenceNumbers
                  (which are greater than or equal NextAck),
                  Retransmit the PDUs on the RetransmissionQ with this range of
                  SequenceNumbers
                Else
                  Do
                Enddo
              Enddo
            Else
              Do
            Enddo
          Enddo
        Else
        Do
          Until the list is exhausted.
        Enddo
      Else
        Do
        Until the list is exhausted.
      Enddo
    Enddo
  Fi
Fi;
If the Ack/Nack List Length ≠ 0 then
If Ack/Nack List flag is True
  Do
    If Sending Ack List Policy Then
      For each pair of Starting and Ending SequenceNumbers,
      Delete this range of PDUs with these SequenceNumbers from the
      RetransmissionQ and discontinue any Retransmission Timer associated
      with them.
    Else
      Do
    Enddo
  Else
    Do
      If Sending Nack List Policy Then
        For each pair of Starting and Ending SequenceNumbers
        (which are greater than or equal NextAck),
        Retransmit the PDUs on the RetransmissionQ with this range of
        SequenceNumbers
      Else
        Do
      Enddo
    Enddo
  Else
    Do
  Enddo
Until the list is exhausted.
Fi

5.8.1.3.2. Formal Specification
To be provided.

5.8.1.4. Syntax
PDU Length: (See Generic Protocol Specification.)
Protocol-id: (See Generic Protocol Specification.)
Protocol-version: (See Generic Protocol Specification.)
Src-ref: (See Generic Protocol Specification.)
Dest-ref: (See Generic Protocol Specification.)
Opcode: (See Generic Protocol Specification.)
Ack/Nack: Integer(SeqNbrLength)
Ack/Nack List Length: Integer(SeqNbrLength)
Ack/Nack List: Sequence(StartingNbr Integer(SeqNbrLength),
                      Ending Integer(SeqNbrLength))

1. If no policy is present, then this is assumed to always evaluate to True.
5.8.1.5. Semantics of the Fields
5.8.1.5.1. PDU Length - See Generic Protocol Specification.


5.8.1.5.5. Dest-ref - See Generic Protocol Specification.

5.8.1.5.6. Opcode - This field distinguishes this PDU as an AckPDU. There are two flags in the opcode field: the Ack/Nack flag indicates whether the Ack/Nack field should be interpreted as Ack or Nack and a Ack/Nack List flag to indicate whether or not the SequenceNumbers are to be Acked or Nacked. The use of the flags is completely independent.

5.8.1.5.7. Ack/Nack - This field contains the base SequenceNumber being Acked or Nacked. If an Ack, then the sender is to assume that no PDUs with SequenceNumbers less than or equal to this one will be requested for retransmission. If a Nack, then the sender is to retransmit all messages from this SequenceNumber to NextSeqToSend-1.

5.8.1.5.8. Ack/Nack List Length - This field specifies the number of Sequence Number pairs in the Ack/Nack List. A value of zero indicates that there is no list present.

5.8.1.5.9. Ack/Nack List - This field consists of pairs of Sequence Numbers. The first designating the beginning of a range of Sequence Numbers and the second the end of the range of Sequence Numbers to be acked or nacked inclusively. This field is not present, if the Ack/Nack List Length is zero (in which case the AckPDU is fixed length).

5.8.2. Retransmission Timer

5.8.2.1. Description
The Retransmission Timer is used to determine when to retransmit PDUs that may have been lost or discarded. The time interval is based on an estimate by the sender of the time to get a positive acknowledgement of the PDU and thus must be coordinated with the Receiving Ack or Ack List Policy.

5.8.2.2. When Activated
This timer is activated when a DataPDU or SetContextPDU is placed in the RetransmissionQ. The expiration time, $T_R$, of this timer is determined by the Sender Error Control Policy.

5.8.2.3. Action Upon Expiration
Upon expiration, the PDUs on the RetransmissionQ are retransmitted to the receiver. A count of retransmissions for the PDUs (or ranges of PDUs) is maintained. The count is incremented after retransmission (either timer or nack) and is cancelled.
Draft

when the PDUs are positively acked. If the count should reach, the ReXmitMax then the Protocol Machine has been unable to maintain the QoS for this connection and an error is indicated.

Note: Certain policies may define timers for the receiver to force retransmission of PDUs or for the source to delete PDUs from the RetransmissionQ even if no positive acknowledgements are received. The behavior of these timers should be defined as part of a policy specification, using the same format used here.
Simple Ack
Receiving Error Control
Policy Specification

1. Introduction
This is a policy for use with the Error Control Mechanism in error control protocols built
upon the Generic Protocol. This Mechanism and Policy extend the function of the Data
Transfer Phase.

2. Policy Definition
This policy defines a simple acknowledgement strategy of sending an Acknowledgement for
every Nth PDU received. The value of N is specified as part of the policy negotiation.

3. Policy Identifier
TBD

4. Description of the Algorithm
The Policy procedure maintains an own local variable that is incremented for each
PDU received. When the value of the variable modulo N is 0 an Ack PDU is sent with the
Sequence Number of the last PDU received.

5. Policy Specification

5.1. Globals Used
5.1.1. Local State
N – The modulus for sending an AckPDU.
NPDU s – The total number of PDUs received on this connection.

5.1.2. PCI
SequenceNumber – (see the Sequencing Mechanism Specification).

5.2. Policy Specification
5.2.1 Specification of Procedures

NPDU s := NPDU s + 1;
If NPDU s mod N = 0 Then
Send an AckPDU on this connection
with the value of the Ack/Nack field set to the value NextSeq-1;
Fi
A Simple Multicast Quorum
Sending Ack Error Control
Policy Specification

1. Introduction
   This Policy specifies a Simple Multicast Quorum Error Control Policy for the basic Unicast
   Error Control Mechanism for an error control protocol. This policy extends the functionality
   of the Data Transfer Phase of the Generic Protocol.

2. Policy Definition
   This policy is designed to work with a multicast error control protocol with a group
   population of M. This policy discards PDUs from the ReTransmissionQ when N of the M
   members of the population of acknowledged the PDU, where N ≤ M-1. This policy is a
   Sending Error Control Policy. The Receiving Error Control Policy is assumed to be one of
   the existing Unicast Error Control Policies.

3. Policy Identifier
   TBD

4. Description of the Algorithm
   Let the Multicast Group consist of M members. There is an implicit pairwise connection
   between the sender and the M-1 receivers. The Source/Destination Reference Numbers
   provide the identification of these M-1 receivers. The Sender maintains a list of which
   members of the Group have acknowledged SequenceNumbers. When N of acknowledged Sequence
   Number all PDUs with SequenceNumbers less than or equal to that Sequence Number are
   deleted from the RetransmissionQ and any RetransmissionTimers associated with those
   PDUs are discontinued. Note that if N = M-1 then an ack is required from all members of the
   group.

5. Policy Specification
   5.1. Globals Used
      5.1.1. Local State
         M – The number of members of the group.
         N – The number of members required for a Quorum.
         X – A Sequence Number value less than or equal to the value of the
             Ack/Nack field of the last AckPDU received.
         Integer Array LastAck[1:M-1] – The Last SequenceNumber acked on
         each pairwise connection.
      5.1.2. PCI
         SequenceNumber – (See the Sequencing Mechanism Specification).

Draft
5.2. Policy Specification

5.2.1 Specification of Procedures

Assign each Source/Destination Reference Number pair a position in the LastAck array.

When an AckPDU is received for a particular Source/Destination Reference Number pair, set the appropriate element of the array to the maximum of the current value of the element and the value of the Ack/Nack field.

When N elements of the array have a value less than or equal to X, delete all PDUs from the ReTransmissionQ with SequenceNumbers less than or equal to X and discontinue any timers associated with the PDUs. Set the value of all elements of the array to X.
Appendix G

Flow Control
Mechanism Specification
and Example Policies
Flow Control
Mechanism Specification

1. Introduction
This document defines Flow Control Mechanisms based on either credits (number of PDUs) or rate (PDUs per unit of time).

2. Facility Provided
The Flow Control Mechanism does not require any other mechanism, although some policies may make use of sequence numbers and state information associated with either the Sequencing or Lost and Duplicate Detection Mechanisms. To facilitate the definition of such policies some of this state information is included here to facilitate specification and understanding.

3. Narrative Description of the Mechanism
The Flow Control Mechanism is a classic feedback flow control scheme. The receiver(s) of PDUs provide an indication to the sender when it can send. The mechanism described here allows two methods of exerting this control: credit, in which the sender is told how many PDUs may be sent before more credit is extended; and rate-based, in which the sender is told at what rate it may emit PDUs. The flow control is always expressed as the number of PDUs or as the number of PDUs per unit time, or more precisely, how many PDUs may be sent in a unit of time. Each approach has its advantages depending on the traffic characteristics of the connection. By having two independent methods, a policy may actually use both. Allowing one to dominate the other, except under extreme conditions when the other may exert an effect. For example, one might allow rate-based flow control to dominate by keeping credit sufficiently high that the receiving transport protocol machine is delivering data at about the same rate. However, the credit level is maintained at the something close to maximum buffer allocation for this connection. If the transport protocol machine should get behind and begin to exceed its maximum buffer allocation, the credit-based flow control would dominate and temporarily stop the flow on the connection, until the buffer pool was restored to an appropriate level.

The flow control is not dependent on having any other mechanisms, such as Sequencing, available. However, certain policies may assume that Sequencing or Lost and Duplicate Detection are available and use state variables from these mechanisms to determine the adjustments in the flow control. (The classic dynamic sliding window is an example of a policy that couples both Sequencing and Lost and Duplicate Detection mechanisms to flow control.) The Flow Control Mechanism only applies to the Data PDUs, not to control PDUs.

4. Mechanism Identifier
Flow Control Mechanism: <to be supplied>
5. Mechanism Specification

5.1. Other Mechanisms Required
None
Note: Since the operation of updating the flow control variables (credit or rate) is replacement, no initialization is required during the allocation phase.

5.2. Imports from other Mechanism Specifications:
None

5.2.1. Local State Variables
Sending
N/A
Receiving
N/A

5.2.2. Extended PDUs
N/A

5.2.3. Other Syntactic Constructs
N/A

5.3. Environmental Variables
N/A

5.4. Additional PCI Required
None

5.4.1. Syntax
N/A

5.4.2. Semantics of the Fields
N/A

5.5. Local State Variables
Sending
SndrCredit – This variable contains the number of PDUs that the sender may send before it must stop sending.
SndrRate – This variable contains the number of PDUs that may be sent in one Time Unit. The rate is defined such that the sender may send the specified number of PDUs in that unit of time. Thus, the rate will not necessarily generate a steady flow, but may exhibit a bursty pattern.
PDUsSentinTimeUnit – This variable contains the number of PDUs sent in this Time Unit. When PDUsSentinTimeUnit equals SndrRate, the sender must wait for the beginning of a new time unit before additional PDUs may be sent.
TimeUnit – This field contains the unit of Time in milliseconds over which the rate is computed.
Receiving
RcvrCredit – This variable contains the number of PDUs that the receiver believes the sender may send before more credit must be extended or the flow on the connection is stopped.

RcvrRate – This variable contains the current rate that the receiver has told the sender that it may send PDUs at.

PDUsRcvdinTimeUnit – This variable contains the number of PDUs received in this Time Unit. When PDUsRcvdinTimeUnit equals RcvrRate, the receiver is allowed to discard any PDUs received until a new time unit begins.

TimeUnit – This field contains the unit of Time in milliseconds over which the rate is computed.

5.6. Additional Actions Associated with the Generic Protocol

5.6.1. Data.submit
5.6.1.1. Informal Specification
This action is invoked after the SDU containing the Data.submit has been fragmented into some number of PDUs. For each resulting PDU:
If SndrCredit ≤ 0 or PDUsSentinTimeUnit ≥ SndrRate Then
   Place the PDU on the WaitingToSendQ;
Else
   SndrCredit := SndrCredit - 1;
   PDUsSentinTimeUnit := PDUsSentinTimeUnit + 1;
   Place the PDU on the SendingQ
Fi

5.6.1.2. Formal Specification

5.6.2. Data PDU
5.6.2.1. Informal Specification
If RcvrCredit ≤ 0 or PDUsRcvdinTimeUnit = RcvrRate Then
   Invoke Flow Control Overrun Policy
Else
   RcvrCredit := RcvrCredit -1;
   PDUsRcvdinTimeUnit := PDUsRcvdinTimeUnit +1;
   Invoke Flow Control Allocation Policy
Fi

5.6.2.2. Formal Specification

5.7. Actions Associated with Imported PDUs
None

5.8. New PDUs or Timers Associated with this Mechanism

5.8.1. New PDU Name
FlowControl PDU
5.8.1.1. Description
This PDU is used to coordinate the flow control among receivers and senders. The PDU is used to deliver credit or rate information to the sender.

5.8.1.2. When Generated
5.8.1.2.1. Informal Specification
This PDU is generated by the Flow Control Allocation Policy when it determines that more credit should be extended to the sender, or when the rate must be adjusted.

5.8.1.3. Action
5.8.1.3.1. Informal Specification
Upon receipt of the Flow Control PDU,
If NewCredit present Then
   SndrCredit := NewCredit
Fi
If NewRate present Then
   SndrRate := NewRate
   TimeUnit := NewTimeUnit
Fi
If WaitingtoSendQ not Empty Then
   Send as many PDUs as possible, given the current allocation
Fi

5.8.1.4. Syntax
PDU Length: (See Generic Protocol Specification.)
Protocol-id: (See Generic Protocol Specification.)
Protocol-version: (See Generic Protocol Specification.)
Src-ref: (See Generic Protocol Specification.)
Dest-ref: (See Generic Protocol Specification.)
Opcode: (See Generic Protocol Specification.)
NewCredit: (CreditLength)
NewRate: (RateLength)
NewTimeUnit: (TimeUnitLength)

5.8.1.5. Semantics of the Fields
NewCredit – This field contains the value of the credit, in terms of the number of PDUs that can be sent, that the receiver is extending to the sender. This value replaces the previous value, so it can, in effect, be used to take back credit.

NewRate – This field contains the value of the rate that the receiver is giving to the sender. The Sender is allowed to send this many PDUs in the given Time Unit.

TimeUnit – This field contains the unit of Time in milliseconds over which the rate is computed.

5.8.2. New Timer Name
None

5.8.2.1. Description
N/A
5.8.2.2. When Activated
N/A

5.8.2.3. Action upon Expiration
N/A
Flow Control Overrun
Policy Specification

1. Introduction

This policy is to be used with the Flow control mechanism built upon the Generic Data Transfer Protocol. This mechanism and policy extend the functionality of both the Allocation and Data transfer phases.

2. Policy Definition

This policy is invoked by the receiver, when the Recvr Credit $\leq 0$ or PDURCVDinTimeUnit = RcvrRate.

3. Policy Identifier

TBD

4. Description of the Algorithm

When the RecvrCredit is $\leq 0$ (i.e. sender sent more PDUS than it is supposed to), this policy simply discards all the additional PDUS until a new time unit begins. Also, when the PDURcvdinTime is equal to RcvrRate, all the PDUs are discarded until a new time unit.

5. Policy Specification

5.1 Specification Of Procedures

if RecvrCredit < 0 then
discard PDU
fi
if PDU RcvdinTimeUnit = RcvrRate then
discard PDU
fi
Flow Control Allocation Policy
Specification

1. Introduction

This policy is to be used with the Flow control mechanism built upon the Generic Protocol. This mechanism and policy extend the functionality of both the Allocation and Data transfer phases.

2. Policy Definition

This policy is invoked by the receiver of the Data PDUs to send credit or rate information.

3. Policy Identifier

TBD

4. Description of the Algorithm

This policy handles two cases. In one case, this policy generates a flow control PDU to inform the sender about the increase of its sendrcredit for sending more PDUs. In the other case, this policy adjusts the sndrrate by generating a flow control PDU.

5. Policy Specification

5.1 Specification Of Procedures

if Newcredit present then
    sndr credit = Newcredit
fi

if Newrate present then
    Sndrrate = Newrate
    TimeUnit = NewTimeUnit
fi
A Simple Multicast Quorum
Sending Flow Control Policy
Specification

1. Introduction

This policy specifies a simple multicast quorum flow control policy for the basic unicast flow control mechanism. This policy extends the functionality of the data transfer phase of the Generic protocol.

2. Policy Definition

This policy is designed to work with a multicast flow control protocol with a group population of M. This policy places N PDUs on the sendingQ, when N of the M members of the population indicate the sender to transmit data PDUs.

3. Policy Identifier

TBD

4. Description of the Algorithm

Let the Multicast group consist of M members. There is an implicit pairwise connection between the sender and M-1 receivers. The sender maintains the list of members of the group that have indicated the sender to transmit PDUs. When N of the members indicated the sender to transmit the PDUs, then the sender removes the corresponding PDU from the waiting to sendQ and places the PDU on sendQ for transmitting to the corresponding receiver.

5. Policy Specification

5.1 Globals used

M -- The number of members in the group
N -- The number of members in a Quorum
Int Lastind[1:M-1]

5.1.2 PCI

SendingQ -- see the flow control mechanism
SendrCredit -- see the flow control mechanism
5.2 Policy Specification

5.2.1 Specification of procedures

Assign each destination a position in the Lastind array.

When an indication is received from a destination, record its number in the lastind array and place the PDU on the waitingtosendQ. When N elements of the array have destination numbers, then remove the PDU from the waitingtosendQ and place it on the corresponding destination's sendingQ for transmitting the PDU to the corresponding receiver. Clear all N elements of the array before the next data transmission starts.
Appendix H

Allocation Mechanism Specification
Allocation Mechanism Specification

1. Introduction

This document specifies the additional mechanism required to provide Allocation and Deallocation phases for a Generic Data Transfer Protocol. The Allocation mechanism also specifies the primitives (Allocate.submit and AllocateRsp.submit) and PDUs (AllocateRqPDU and AllocateRspPDU) for Allocation and primitives (Dealocate.submit and DeallocateRsp.submit) and PDUs(DeallocatePDU) for Deallcation.

2. Facility Provided

This mechanism is used to establish shared state that requires the exchange of protocol for the coordination of the mechanisms to be employed.

3. Narrative Description of the Mechanism

The Allocation mechanism provides each communicating peer, the means to establish sufficient shared state for the exchange of data. When a service user wishes to establish a communication, it invokes the Allocate.submit primitive with appropriate parameters to notify the protocol machine of its request for communication resources. The protocol machine, then determines whether this requires a two-way handshake or a three-way handshake and sends a AllocateRqPDU to its peer. The peer protocol machine informs its acceptance of the communication request by invoking the AllocateRsp.submit primitive and sending an AllocateRspPDU. For communications using two-way handshake, this completes the Allocation phase. Communications that require three-way handshake, the Allocation phase is not complete until the AllocateCnfPDU is received by the communicating peer that initiated the request for communication. The service user can terminate the communication by issuing the Dealocate.submit primitive and sending an DeallocatePDU.

3.1 Policies

The following policies are used with the Allocation mechanism:

3.1.1 Receiver New Connection Policy: This policy is used by the receiver of an AllocateRqPDU and is used to determine whether the layer has the resources to accept another connection.

3.1.2 Sender Allocation Policy: This policy is used to moderate when the initiating service user is notified of the success or failure of a connection request. In case of a three-way handshake variation of the allocation mechanism, this policy moderates when the protocol machine sends AllocateCnfPDU.

3.1.3 Sender Deallcation Policy: This policy is used to determine when the service user received the Dealocate.submit primitive.

4. Mechanism Identifier

To be assigned.
5. Mechanism Specification

5.1 Other Mechanisms required

This Mechanism assumes only the Generic Data Transfer Protocol

5.2 Imports From Other Mechanism Specifications

This mechanism does not import other variables from other Mechanisms.

5.3 Additional Actions Associated With the Generic Protocol

5.3.1 Allocate.submit

5.3.1.1 Informal Specification
If the protocol state machine is in NULL state then
  enters the Pending state
  determine whether this requires a two-way or three-way handshake
  send an AllocateRQPDU using the Allocate.submit primitive parameters
  set the Allocation Timer(TA)
else
  This is a protocol error
Fi

5.3.1.2 Formal Specification
To Be Provided

5.3.2 AllocateRsp.submit

5.3.2.1 Informal Specification
If the protocol state machine is in the PENDING state then
  send an AllocateRspPDU using the
  AllocateRsp.primitive parameters to the destination address
  (which was the source address in the AllocateRQPDU)
If this is a Two-Way Handshake allocation then
  the protocol state machine enters the CONNECTED state
else
  the protocol state machine enters the PENDINGCNFM state
Fi
else
  This is a protocol error
Fi

5.3.2.2 Formal Specification
To Be Provided

5.3.3 AllocateRQPDU

5.3.1.1 Informal Specification
If protocol state machine is in NULL state then
  If Receiving new connection policy then
    Create local State Vector for exchange with this sender. The recipient protocol machine
    creates a new instantiation of state, i.e. a connection, and transitions to the pending state.
  If using Receiver Allocate policy then
    Invoke Allocate.deliver
Fi
else
    Request is not acceptable at this time
    send DeallocatePDU using AllocateRQPDU
    parameters and remain in the NULL state
Fi
else
    If protocol state machine is in PENDING state then
    send an AllocateRspPDU to the sender
else
    Notify the local system of an illegal PDU
Fi
Fi

5.3.1.2 Formal Specification
To Be Provided

5.3.4 AllocateRspPDU
5.3.1.1 Informal Specification
If the protocol state machine is in PENDING state then
    cancel Allocation Timer(TA)
    if senderAllocatePolicy then
        notify service user that allocation was successful through Allocate.deliver primitive
        Protocol state machine will transition to the CONNECTED state
    If this is a Three-Way Handshake then
        send the AllocateCnfPDU
    Fi
    Fi
else
    This is a protocol error
Fi

5.3.1.2 Formal Specification
To Be Provided

5.3.5 AllocateCnfPDU
5.3.5.1 Informal Specification
If protocol state machine is in PENDINGCNFM state then
    it transitions to the CONNECTED state
else
    If the protocol state machine is any other state then
        This is a protocol error
    Fi

5.3.5.2 Formal Specification
5.3.6 Deallocate.submit
5.3.6.1 Informal Specification
If protocol state machine is in the connected state then
    discard any PDUs queued for sending
    sends a DeallocatePDU
    Initiates Deallocation Timer,TD
    transitions to the DEALLOCPENDING state
else
If the protocol state machine is in DeallocatePending or allocatePending then send another Deallocate PDU else invoke Deallocate.deliver Fin

5.3.6.2 Formal Specification
To Be Provided

5.3.7 DeallocateRsp.submit
5.3.7.1 Informal Specification
If the protocol state machine receives a DeallocateRsp.submit primitive then transition to the NULL state Fin

5.3.1.2 Formal Specification
To Be Provided

5.3.8 DeallocatePDU
5.3.8.1 Informal Specification
If the protocol state machine is in the NULL state then send a DeallocatePDU remain in the NULL state else If the protocol state machine is in the DeAllocPENDING state then Cancel TD send a DeallocatePDU Transition to the NULL state If sending Deallocation Policy then invoke DeallocateRsp.deliver Fin else If Receiver Deallocation Policy then invoke a Deallocate.deliver primitive Fin

5.3.8.2 Formal Specification
To Be Provided

5.4 New PDUs Or Timers Associated with this Mechanism
5.4.1 Allocation Timer
5.4.1.1 Description
The Allocation Timer is used to determine when to retransmit AllocatePDUs that may have been lost or discarded. The time interval is based on an estimate by the sender of the time to get a AllocateRspPDU and thus must be coordinated with the Receiving allocation policy.

5.4.1.2 When Generated
This timer is activated when a AllocatePDU is sent. The expiration time of this timer is determined by the sender Allocation Policy.