Network Working Group
Request for Comments: 1301

S. Armstrong
Xerox
A. Freier
Apple
K. Marzullo
Cornell

February 1992

Multicast Transport Protocol

Status of this Memo

This memo provides information for the Internet community. It does not specify an Internet standard. Distribution of this memo is unlimited.

Summary

This memo describes a protocol for reliable transport that utilizes the multicast capability of applicable lower layer networking architectures. The transport definition permits an arbitrary number of transport providers to perform realtime collaborations without requiring networking clients (aka, applications) to possess detailed knowledge of the population or geographical dispersion of the participating members. It is not network architectural specific, but does implicitly require some form of multicasting (or broadcasting) at the data link level, as well as some means of communicating that capability up through the layers to the transport.

Keywords: reliable transport, multicast, broadcast, collaboration, networking.

Table of Contents

1. Introduction

This document describes a flow controlled, atomic multicasting transport protocol (MTP). The purpose of this document is to present sufficient information to implement the protocol.

The MTP design has been influenced by the large body of the networking and distributed systems literature and technology that has been introduced during the last decade and a half. Representative sources include [Xer81], [BSTM79] and [Pos81] for transport design, and [Bog83] and [DIX82] for general concepts of broadcast and multicast. [CLZ87] influenced MTP's retransmission mechanisms, and [Fre84] influenced the transport timings. MTP over IP uses mechanisms

described in [Dee89]. MTP's ordering and agreement protocols were influenced by work done in [CM87], [JB89] and [Cri88]. Finally, a description of MTP's philosophy and its motivation can be found in [AFM91].

2. Protocol description

MTP is a transport in that it is a client of the network layer (as defined by the OSI networking model) [1]. MTP provides reliable delivery of client data between one or more communicating processes, as well as a predefined principal process. The collection of processes is called a web.

In addition to transporting data reliably and efficiently, MTP provides the synchronization necessary for web members to agree on the order of receipt of all messages and can agree on the delivery of the message even in the face of partitions. This ordering and agreement protocol uses serialized tokens granted by the master to producers.

The processes may have any one of three levels of capability. One member must be the master. The master instantiates and controls the behavior of the web, including its membership and performance. Non master members may be either producer/consumers or pure consumers. The former class of member is permitted to transmit user data to the entire membership (and expected to logically hear itself), while the latter is prohibited from transmitting user data.

MTP is a negative acknowledgement protocol, exploiting the highly reliable delivery of the local area and wide area network technologies of today. Successful delivery of data is accepted by consuming stations silently rather than having the successful delivery noted to the producing process, thus reducing the amount of reverse traffic required to maintain synchronization.

2.1 Definition of terms

The following terms are used throughout this document. They are defined here to eliminate ambiguity.

consumer — A consumer is a transport that is capable only of receiving user data. It may transmit control packets, such as negative acknowledgements, but may never transmit any requests for the transmit token or any form of data or empty messages.

heartbeat — A heartbeat is an interval of time, nominally measured in milliseconds. It is a key parameter in the transport's

state and can be adapted to the requirements of the transport's client to provide the desired quality of service.

master — The master is the principal member of the web. The master capability is a superset of a producer member. The master is mainly responsible for giving out transmit tokens to members who wish to send data, and overseeing the web's membership and operational parameters.

member — A web member is any process that has been permitted to join the web (by the master) as well as the master itself.

membership class — Every member is classified as to its intentions for joining the web. Membership classes are defined to be consumer, producer and master. Each successive class is a formal superset of the previous.

message — An MTP message is a concatenation of the user data portions of a series of data packets with the last packet in the series carrying an end of message indication. A message may contain any number of bytes of user data, including zero.

NSAP — The network service access point. This is the network address, or the node address of the machine, where a service is available.

producer — Producer is a class of membership that is a formal superset of a consumer. A producer is permitted (and expected) to transmit client data as well as consume data transmitted by other producers.

retention — Retention is one of the three fundamental parameters that make up the transport's state (along with heartbeat and window). Retention is a number of heartbeats, and though applied in several different circumstances, is primarily used as the number of heartbeats a producing client must maintain buffered data should it need to be retransmitted.

token — In order to transmit, a producer must first be in possesion of a token. Tokens are granted only by the master and include the message sequence number. Consequently, they are fundamental in the operation of the ordering and agreement protocol used by MTP.

TSAP — The transport service access point. This is the address that uniquely defines particular instantiation of a service. TSAPs are formed by logically concatenating the node's NSAP with a transport identifier (and perhaps a packet/protocol type).

user data — User data is the client information carried in MTP data packets and treated as uninterpreted octets by the transport. The end of message and subchannel indicators are also be treated as user data.

web — A collection of processes collaborating on the solution of a single problem.

window — The window is one of the fundamental elements of the transport's state that can be controlled to affect the quality of service being provided to the client. It represents the number of user data carrying packets that may be multicast into the web during a heartbeat by a single member.

2.2 Packet format

An MTP packet consists of a transport protocol header followed by a variable amount of data. The protocol header, shown in Figure 1, is part of every packet. The remainder of the packet is either user data (packet type = data) or additional transport specific information. The fields in the header are statically defined as n-bit wide quantities. There are no undefined fields or fields that may at any time have undefined values. Reserved fields, if they exist, must always have a value of zero.

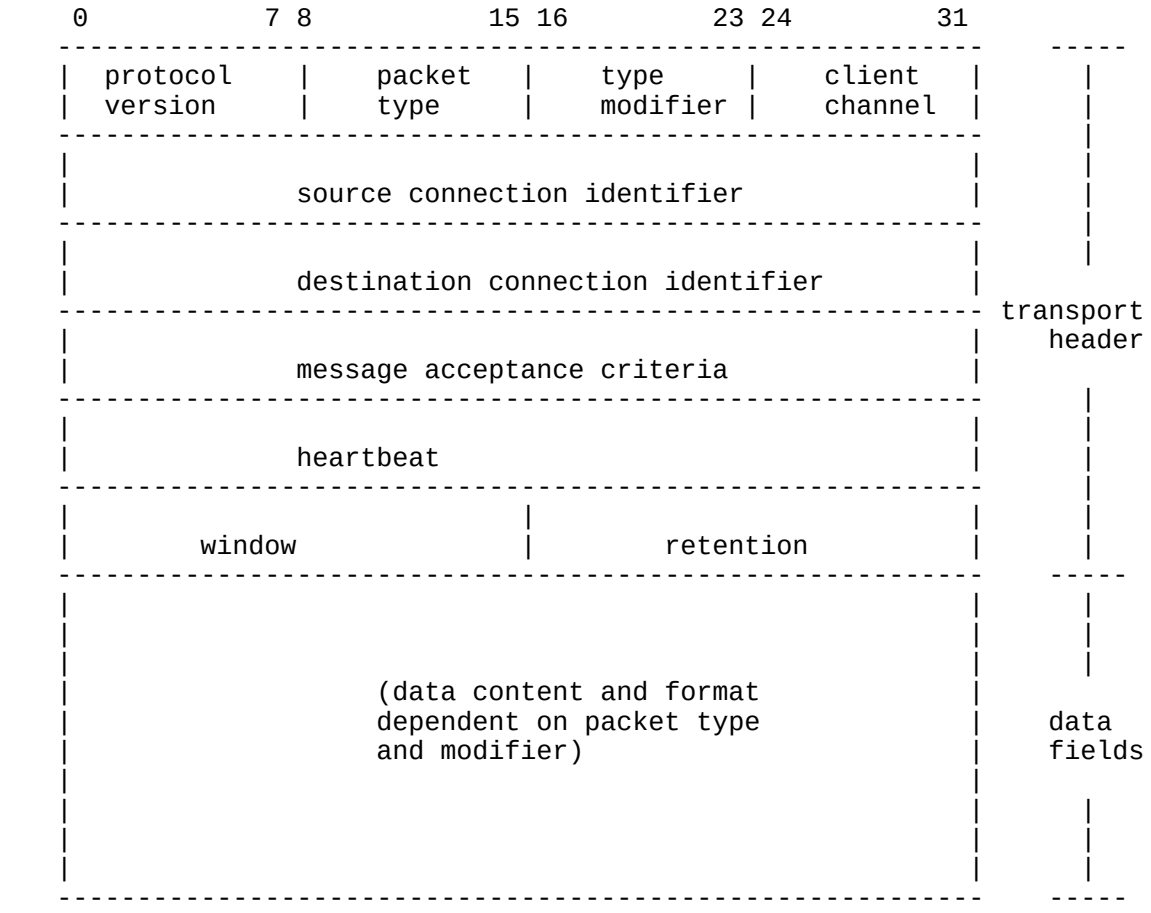

Figure 1. MTP packet format

2.2.1. Protocol version

The first 8 bits of the packet are the protocol version number. This document describes version 1 of the Multicast Transport Protocol and thus the version field has a value of 0x01.

2.2.2. Packet type and modifier

The second byte of the header is the packet type and the following byte contains the packet type modifier. Typical control message exchanges are in a request/response pair. The modifier field simplifies the construction of responses by permitting reuse of the incoming message with minimal modification. The following table gives the packet type field values along with their modifiers. The modifiers are valid only in the context of the type. In the prose of the definitions and later in the document, the syntax for referring to one of the entries described in the following table will be type[modifier]. For example, a reference to data[eow] would be a packet of type data with an end of window modifier.

type	modifier	description
data(0)	data(0)	The packet is one that contains user information. Only the process possessing a transmit token is permitted to send data unless specifically requested to retransmit previously transmitted data. All packets of type data are multicast to the entire web.
	eow(1)	A data packet with the eow (end of window) modifier set indicates that the transmitter intends to send no more packets in this heartbeat either because it has sent as many as permitted given the window parameter or simply has no more data to send during the current heartbeat. This is not client information but rather a hint to be used by transport providers to synchronize the computation and transmission of naks.
	eom(2)	Data[eom] marks the end of the message to the consumers, and the surrendering of the transmit token to the master. And like a data[eow] a data[eom] packet implies the end of window.
nak(1)	request(0)	A nak[request] packet is a consumer requesting a retransmission of one or more

		data packets. The data field contains an ordered list of packet sequence numbers that are being requested. Naks of any form are always unicast.
	deny(1)	A nak[deny] message indicates that the producer source of the nak[deny]) cannot retransmit one or more of the packets requested. The process receiving the nak[deny] must report the failure to its client.
empty(2)	dally(0)	An empty[dally] packet is multicast to maintain synchronization when no client data is available.
	cancel(1)	If a producer finds itself in possession of a transmit token and has no data to send, it may cancel the token[request] by multicasting an empty[cancel] message.
	hibernate(2)	If the master possesses all of the web's transmit tokens and all outstanding messages have been accepted or rejected, the master may transmit empty[hibernate] packets at a rate significantly slower than indicated by the web's value of heartbeat.
join(3)	request(0)	A join[request] packet is sent by a process wishing to join a web to the web's unknown TSAP (see section 2.2.5).
	confirm(1)	The join[confirm] packet is the master's confirmation of the destination's request to join the web. It will be unicast by the master (and only the master) to the station that sent the join[request].
	deny(2)	A join[deny] packet indicates permission to join the web was denied. It may only be transmitted by the master and will be unicast to the member that sent the join[request].
quit(4)	request(0)	A quit[request] may be unicast to the master by any member of the web at any time to indicate the sending process wishes to withdraw from the web. Any member may unicast a quit to another member requesting that the

		destination member quit the web due to intolerable behavior. The master may multicast a quit[request] requiring that the entire web disband. The request will be multicast at regular heartbeat intervals until there are no responses to retention requests.
	confirm(1)	The quit[confirm] packet is the indication that a quit[request] has been observed and appropriate local action has been taken. Quit[confirm] are always unicast.
token(5)	request(0)	A token[request] is a producing member requesting a transmit token from the master. Such packets are unicast to the master.
	confirm(1)	The token[confirm] packet is sent by the master to assign the transmit token to a member that has requested it. token[confirm] will be unicast to the member being granted the token.
isMember(6)	request(0)	An isMember[request] is soliciting verification that the target member is a recognized member of the web. All forms of the isMember packet are unicast to a specific member.
	confirm(1)	IsMember[confirm] packets are positive responses to isMember[requests].
	deny(2)	If the member receiving the isMember[request] cannot confirm the target's membership in the web, it responds with a isMember[deny].

2.2.3. Subchannel

The fourth byte of the transport header contains the client's subchannel value. The default value of the subchannel field is zero. Semantics of the subchannel value are defined by the transport client and therefore are only applicable to packets of type data. All other packet types must have a subchannel value of zero.

2.2.4. Source connection identifier

The source connection identifier field is a 32 bit field containing a transmitting system unique value assigned at the time the transport

is created. The field is used in identifying the particular transport instantiation and is a component of the TSAP. Every packet transmitted by the transport must have this field set.

2.2.5. Destination connection identifier

The destination connection identifier is the 32 bit identifier of the target transport. From the point of view of a process sending a packet, there are three types of destination connection identifiers. First, there is the unknown connection identifier (0x00000000). The unknown value is used only as the destination connection identifier in the join[request] packet.

Second, there is the multicast connection identifier gleaned from the join[confirm] message sent by the master. The multicast connection identifier is used in conjunction with the multicast NSAP to form the destination TSAP of all packets multicast to the entire web [2].

The last class of connection identifier is a unicast identifier and is used to form the destination TSAP when unicasting packets to individual members. Every member of the web has associated with it a unicast connection identifier that is used to form its own unicast TSAP.

2.2.6. Message acceptance

MTP ensures that all processes agree on which messages are accepted and in what order they are accepted. The master controls this aspect of the protocol by controlling allocation of transmit tokens and setting the status of messages. Once a token for a message has been assigned (see section 3.2.1) the master sets the status of that message according to the following rules [AFM91]:

If the master has seen the entire message (i.e., has seen the data[eom] and all intervening data packets), the status is accepted.

If the master has not seen the entire message but believes the message sender is still operational and connected to the master (as determined by the master), the status is pending.

If the master has not seen the entire message and believes the sender to have failed or partitioned away, the status is rejected.

Message status is carried in the message acceptance record (see Figure 2) of every packet, and processes learn the status of earlier messages by processing this information.

The acceptance criteria is a multiple part record that carries the

rules of agreement to determine the message acceptance. The most significant 8 bits is a flag that, if not zero, indicates synchronization is required. The field may vary on a per message basis as directed by producing transport's client. The default is that no synchronization is required.

The second part of the record is a 12 element vector that represents the status of the last 12 messages transmitted into the web.

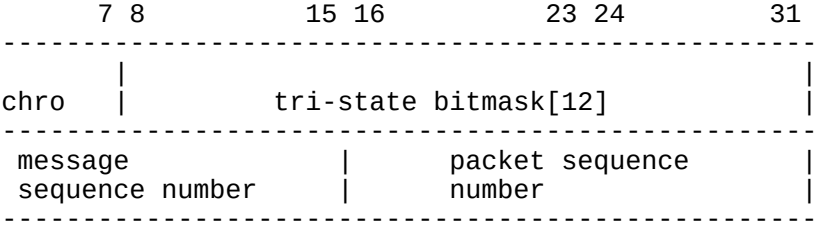

Figure 2. Message acceptance record

Each element of the array is two bits in length and may have one of three values: accepted(0), pending(1) or rejected(2). Initially, the bit mask is set to all zeros. When the token for message m is transmitted, the first (left-most) element of the vector represents the the state of message m - 1, the second element of the vector is the status of message m - 2, and so forth. Therefore the status of the last 12 messages are visible, the status of older messages are lost, logically by shifting the elements out of the vector. Only the master is permitted to set the status of messages. The master is not permitted to shift a status of pending beyond the end of the vector. If that situation arises, the master must instead not confirm any token[request] until the oldest message can be marked as either rejected or accepted.

Message sequence numbers are 16 bit unsigned values. The field is initialized to zero by the master when the transport is initialized, and incremented by one after each token is granted. Only the master is permitted to change the value of the message sequence number. Once granted, that message sequence number is consumed and the state of the message must eventually become either accepted or rejected. No transmit tokens may be granted if the assignment of a message sequence number that would cause a value of pending to be shifted beyond the end of the status vector.

Packet sequence numbers are unsigned 16 bit numbers assigned by the producing process on a per message basis. Packet sequence numbers start at a value of zero for each new message and are incremented by one (consumed) for each data packet making up the message. Consumers

detecting missing packet sequence numbers must send a nak[request] to the appropriate producer to recover the missed data.

Control packets always contain the message acceptance criteria with a synchronization flag set to zero (0x00), the highest message sequence number observed and a packet sequence number one greater than previously observed. Control packets do not consume any sequence numbers. Since control messages are not reliably delivered, the acceptance criteria should only be checked to see if they fall within the proper range of message numbers, relative to the current message number of the receiving station. The range of acceptable sequence numbers should be m-11 to m-13, inclusive, where m is the current message number.

2.2.7. Heartbeat

Heartbeat is an unsigned 32 bit field that has the units of milliseconds. The value of heartbeat is shared by all members of the web. By definition at least one packet (either data, empty or quit from the master) will be multicast into the web within every heartbeat period.

2.2.8. Window

The allocation window (or simply window) is a 16 bit unsigned field that indicates the maximum number of data packets that can be multicasted by a member in a single heartbeat. It is the sum of the retransmitted and new data packets.

2.2.9. Retention

The retention field is a 16 bit unsigned value that is the number of heartbeats for which a producer must retain transmitted client data and state for the purpose of retransmission.

2.3 Transport addresses

Associated with each transport are logically three transport service access points (TSAP), logically formed by the concatenation of a network service access point (NSAP) and a transport connection identifier. These TSAPs are the unknown TSAP, the web’s multicast TSAP and each individual member’s TSAP.

2.3.1. Unknown transport address

Stations that are just joining must use the multicast NSAP associated with the transport, but are not yet aware of either the web’s multicast TSAP the master process’ TSAP. Therefore, joining stations

fabricate a temporary TSAP (referred to as a unknown TSAP) by using a connection identifier reserved to mean unknown (0x00000000). The join[confirm] message will be sourced from the master's TSAP and will include the multicast transport connection identifier in the data field. Those values must be extracted from the join[confirm] and remembered by the joining process.

2.3.2. Web's multicast address

The multicast TSAP is formed by logically concatenating the multicast NSAP associated with the transport creation and the transport connection identifier returned in the data field of the join[confirm] packet. If more than one network is involved in the web, then the multicast transport address becomes a list, one for each network represented. This list is supplied in the data field of token[confirm] packets.

The multicast TSAP is used as the target for all messages that are destined to the entire web, such as data and empty. The master's decision to abandon the transport (quit) is also sent to the multicast transport address.

2.3.3. Member addresses

The member TSAP is formed by using the process' unicast NSAP concatenated with a locally generated unique connection identifier. That TSAP must be the source of every packet transmitted by the process, regardless of its destination, for the lifetime of the transport.

Packets unicast to specific members must contain the appropriate TSAP. For producers and consumers this is not difficult. The only TSAPs of interest are the master and the station(s) currently transmitting data.

3. Protocol behavior

This section defines the expectations of the protocol implementation. These expectations should not be considered guidelines or hints, but rather part the protocol.

3.1 Establishing a transport

Before any rendezvous can be affected, a process must first acquire an NSAP that will be the service access point for the instantiation [3]. The process that first establishes at that NSAP is referred to as the master of the web. The decision as to what process acts as the master must be made a priori in order to guarantee unambiguous

creation in the face of network partitions. The process should make a robust effort to verify that the NSAP being used is not already in service. It may do so by repeatedly sending join[requests] to the web's unknown TSAP. If there is no response to repeated transmissions the process may be relatively confident that the NSAP is not in use and proceed with the creation of the web. If not, the creation must be aborted and the situation reported to its client.

3.1.1. Join request

Additional members may join the web at any time after the establishment of the master by the joining process sending a join[request] to the unknown TSAP. The joining process should have already assigned a unique connection identifier to its transport instantiation that will be used in the source TSAP of the join[request]. The join[request] must contain zeros in all of the acceptance fields. The heartbeat, window and retention parameters are filled in as requested by the transport provider's client. The data of the message must contain the type, class and quality of service parameters that the client has requested.

field	class	definition
membership class	master(0)	There can be only a single web master, and that member has all privileges of a producer class member plus those acquitted only to the master.
	producer(1)	A process that has producer class membership wishes to transmit data into the web as well as consume.
	consumer(2)	A consumer process is a read only process. It will send naks in order to reliably receive data but will never ask for or be permitted to take possession of a transmit token.
transport class	reliable(0)	Specifies a reliable transport, i.e., one that will generate and process naks. The implication is that the data will be reliably delivered or the failure will be detected and reported to the client.
	unreliable(1)	The transport supports best

		effort delivery. Such a transport may still fail if the error rates are too high, but tolerable loss or corruption of data will be permitted [4].
transport type	NxN(0)	The transport will accept multiple processes with producing capability.
	1xN(1)	A 1xN transport permits only a single producer whose identity was established a priori.

The client's desire for minimum throughput (expressed in kilobytes per second) is the lowest value that will be accepted. That throughput is calculated using the heartbeat and window parameters of the transport, and the maximum data unit size, not by measuring actual traffic. Any member that suggests a combination of those parameters that result in an unacceptable throughput will be ignored or asked to withdraw from the web.

A joining client may also suggest a maximum data unit size. This field is expressed as a number of bytes that can be included in a data packet as client data.

If no response is received in a single heartbeat, the join[request] should be retransmitted using the same source TSAP so the master can detect the difference between a new process and a retransmission of a join[request].

3.1.2. Join confirm/deny

Only the master of the web will respond to join[request]. The response may either permit the entry of the new process or deny it. The request to join may be denied because the new member is specifying service parameters that are in conflict with those established by the master. If the join is confirmed the join[confirm] will be unicast by the master with a data field that contains the web's current operating parameters. If those parameters are unacceptable to the joining process it may decide to withdraw from the web. Otherwise the parameters must be accepted as the current operating values.

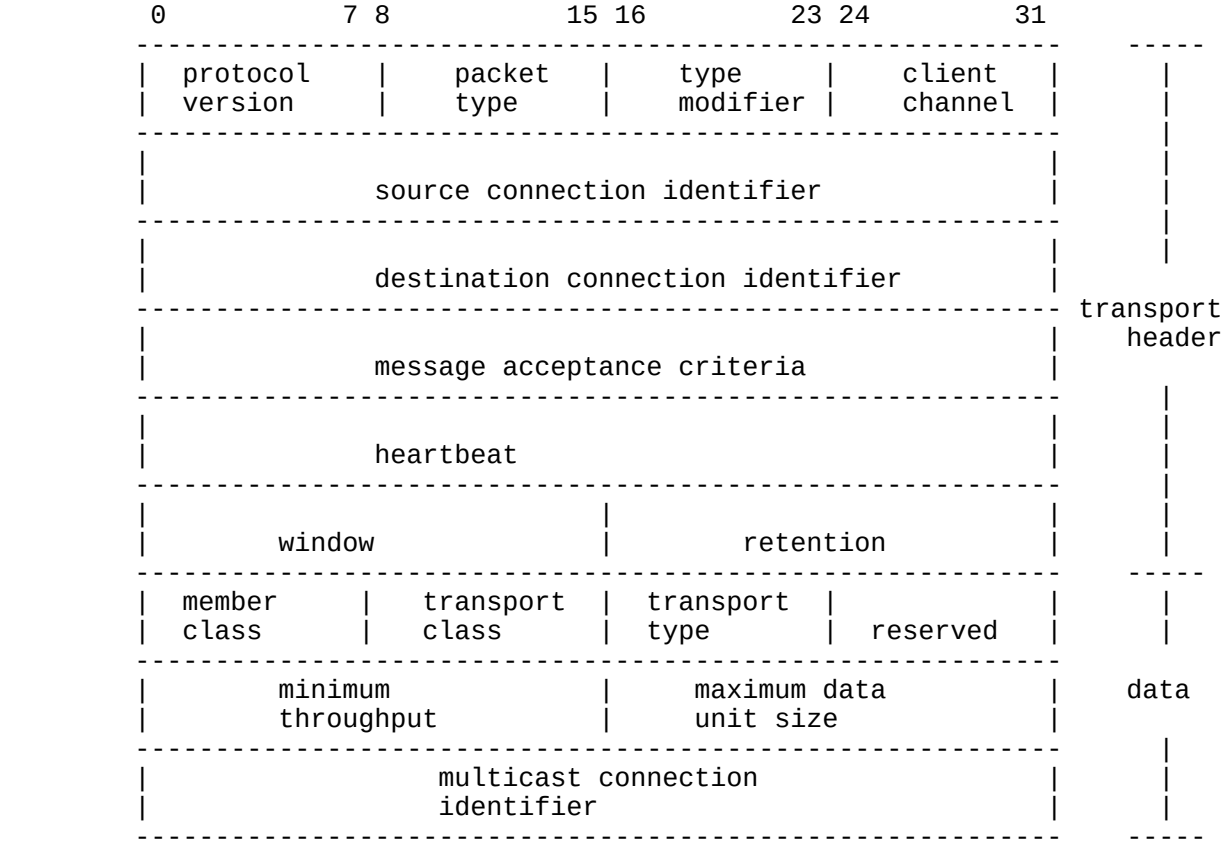

Figure 3. join packet

The join[confirm] will also contain the multicast connection identifier. This must be used to form the TSAP that will be the destination for all multicast messages for the transport. The source

of the join[confirm] message will be the master's TSAP and must be recorded by the member for later use.

The master must be in possession of all the transmit tokens when it sends a join[confirm]. Requiring the master to have the transmit tokens insures that the joining member will enter the web and observe only complete messages. It also permits a notification of the master's client of the join so that application state may be automatically sent to the newly joining member. The newly joined member may be on a network not previously represented in the web's membership, thus requiring a new multicast TSAP be added to the existing list. The entire list will be conveyed in the data field of all subsequent token[confirm] messages (described later).

3.2 Maintaining data consistency

The transport is responsible for maintaining the consistency of the data submitted for delivery by producing clients. The actual client data, while representing the bulk of the information that flows through the web, is accompanied by significant amounts of protocol state information. In addition to the state information piggybacked with the client data, there is a minimum amount of protocol packets that are purely for use by the transport, invisible to the transport client.

3.2.1. Transmit tokens

Before any process may transmit client data or state it must first possess a transmit token. It may acquire the token by transmitting a token[request] to the master. Requests should be unicast to the master's TSAP and should be retransmitted at intervals approximately equal to the heartbeat. Since it is the central source for a transmit token, the master may apply some fairness algorithms to the passing of permission to transmit. At a minimum the requests should be queued in a first in, first out order. Duplicate requests from a single member should be ignored, keeping instead the first unhonored request. When appropriate, the master will send a member with a request pending a token[confirm]. The data field of the response contains all the multicast TSAPs that are represented in the current web at that point in time.

If the master detects no data or heartbeat messages being transmitted into the web it will assume the token is lost, presumably because the member holding the token has failed or has become partitioned away from the master. In such cases, the master may attempt to confirm the state of the process (perhaps by sending isMember[request]). If the member does not respond it is removed from the active members of the web, the message is marked as rejected, the token is assumed by the

master.

Figure 4 shows a timing diagram of a token pass. Increasing time is towards the bottom of the figure. In this figure, process A has a token, and process B requests a token when there are no free tokens.

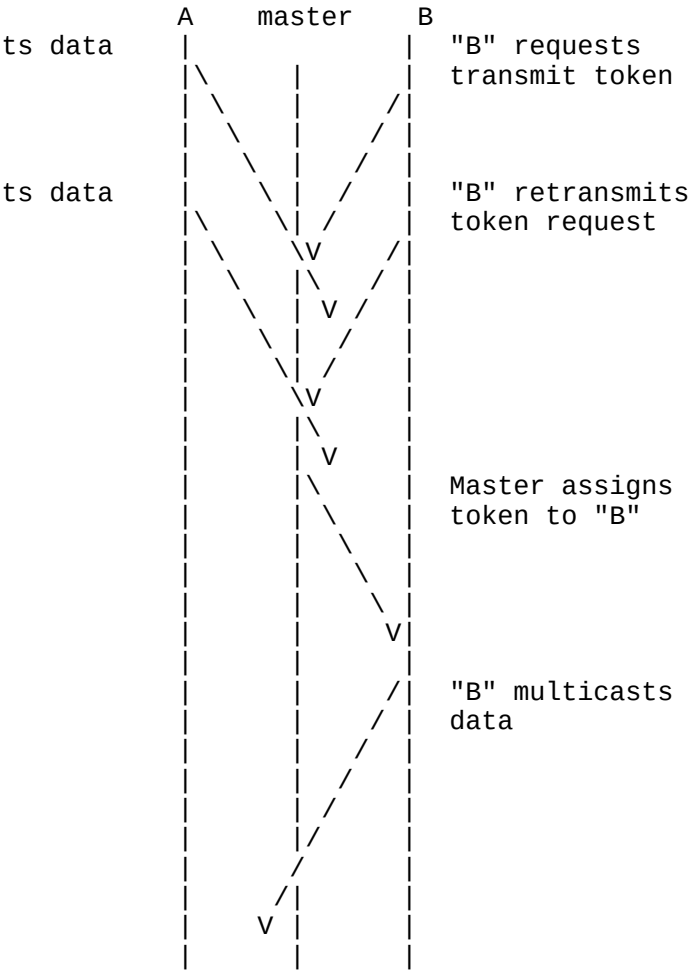

Figure 4. Acquiring the token

Token packets, like other control packets, do not consume sequence numbers. Hence, the master must be able to use another mechanism to determine whether multiple token[request] from a single member are actually requests for a separate token, or are a retransmission of a token[request]. To carry out this obligation, the master and the members must have an implicit understanding of each other’s state.

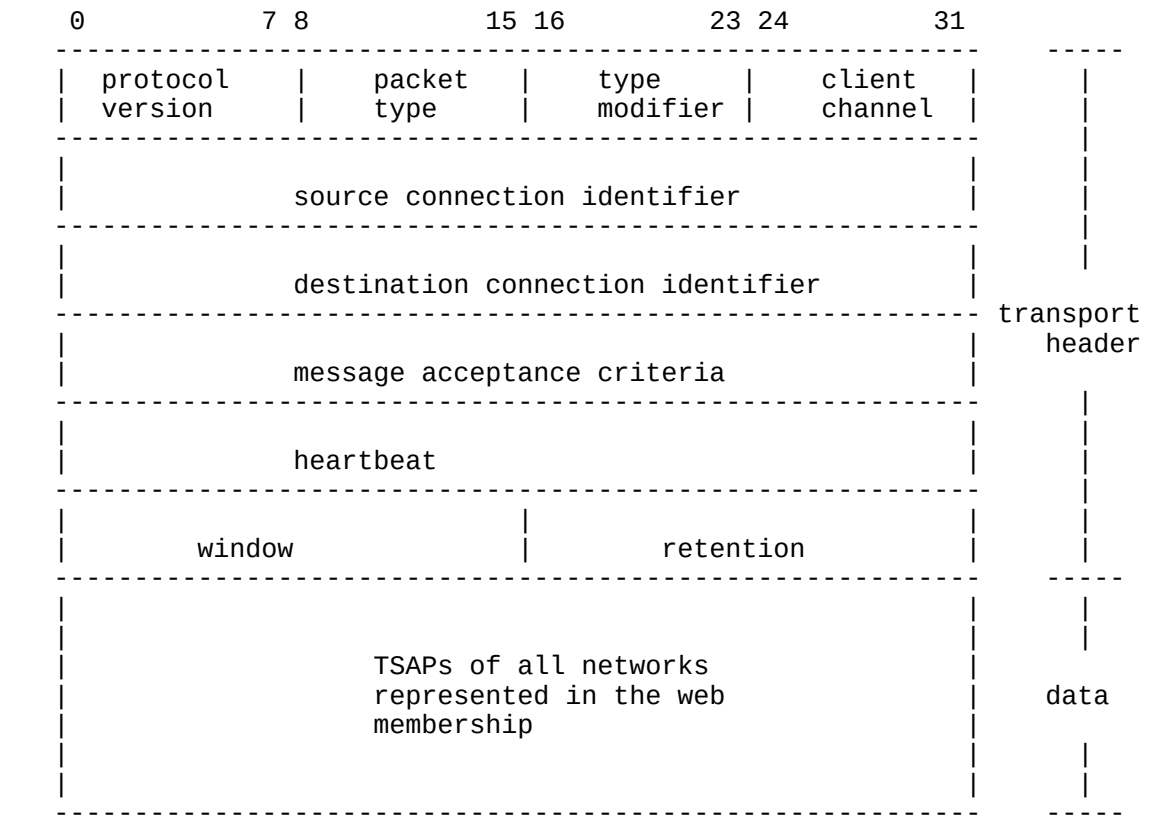

Figure 5. token packet

Assume that the token, as viewed by the master, has three states:

idle The token is not currently assigned. Specifically the message number that it defines is not represented in the current message acceptance vector.

pending The token has been assigned by the master via a token[confirm] packet, but the master has not yet seen any data packets to indicate that the from the producing member received the notification.

busy The token has been assigned and the master has seen data packets carrying the assigned message number. The message comprised by those packets is still represented in the message acceptance vector.

Furthermore, a token that is not idle also has associated with its

state the TSAP of the process that owns (or owned) the token.

Based on this state, the master will respond to any process that has a token in pending state with a reassignment of that token. This is based on the assumption that the original token[confirm] was not received by the requesting process. The only other possibility is that the process did receive the token and transmitted data packets using that token, but the master did not see them. But data messages are by design multi-packet messages, padded with empty packets if necessary. The possibility of the master missing all of the packets of a message is considered less than the possibility of the requesting process missing a single token[confirm] packet.

The process requesting tokens must consider the actions of the master and what prompted them. In most cases the assumptions made by the master will be correct. However, there are two ambiguous situations. There is the situation that the master is most directly addressing, not knowing whether the requesting process has failed to observe the token[confirm] or the master has failed to see data packets transmitted by the producing process. There is also the possibility that the requesting process timed out too quickly and the retransmission of the token[request] passed the token[confirm] in the night. In any case the producing process may find itself in possession of a token for which it has no need. These can be dismissed by sending an empty[cancel] packet.

Another possibility is that the requesting process has actually made use of the assigned token and is requesting another token. Unless the master has observed data using the token, the master will still consider the token pending. Therefore, a process that receives a duplicate token[confirm] should interpret it as a nak and retransmit any data packets previously sent using the token’s message sequence number.

3.2.2. Data transmission

Data is provided by the transport client in the form of uninterpreted bytes. The bytes are encapsulated in packets immediately following the protocol’s fixed overhead fields. The packet may have any number of data bytes between zero and the maximum number of bytes of a network protocol packet minus the network overhead and the fixed transport overhead. Every packet that consumes a sequence number must contain either client data or client state transitions such as the end of message indicator or a subchannel transition.

Packets are transmitted in bursts of packets called windows. The protocol guarantees that no more than the current value of window data packets will be transmitted by a single process during a

heartbeat. Every packet transmitted always contains the latest heartbeat, window and retention information. If full packets are unavailable [5], empty[dally] messages should be transmitted instead. The only packets that will be transmitted containing less than maximum capacity will be data[eom] or those containing client subchannel transitions.

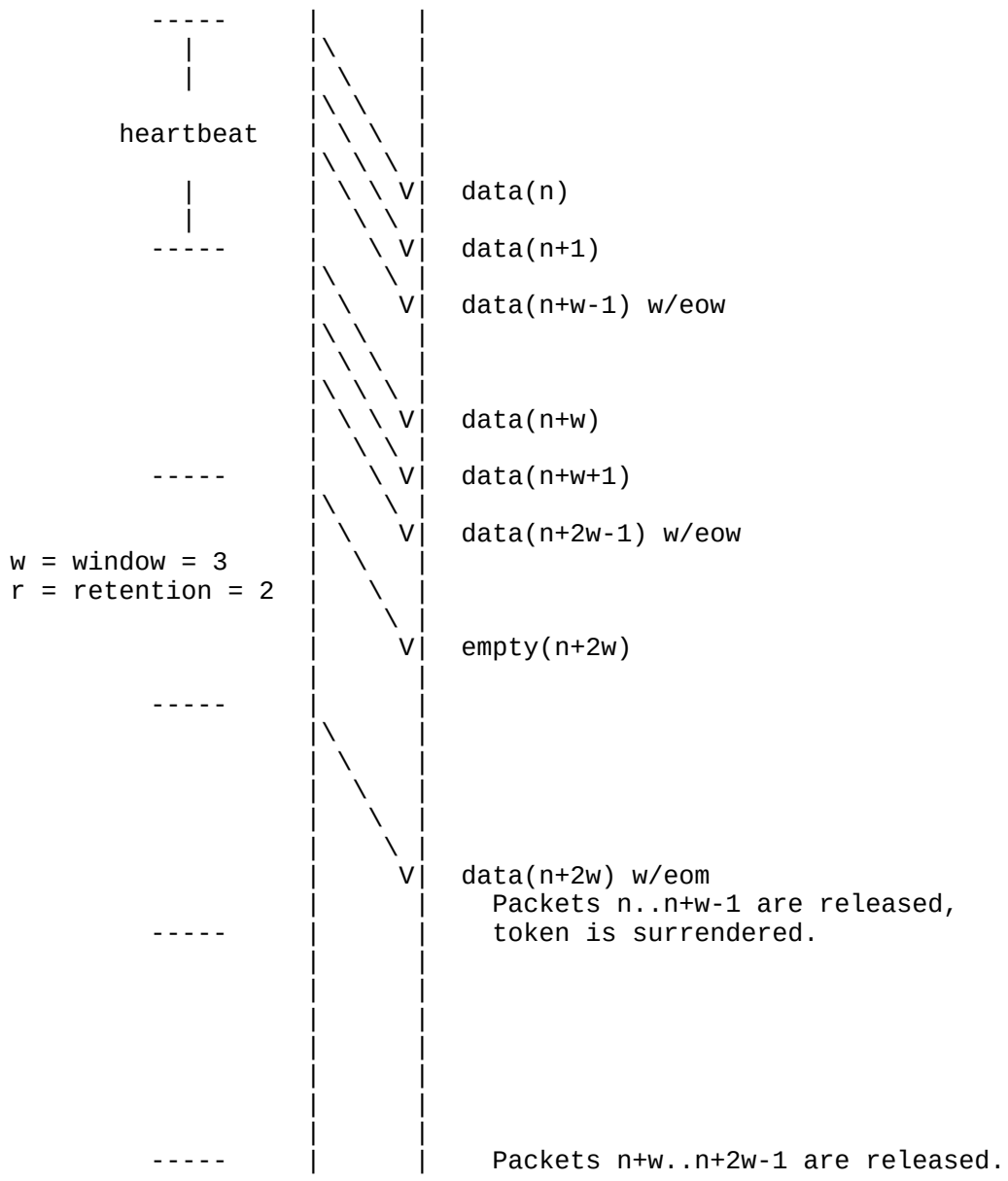

Figure 6. Normal data transmission

Figure 6 shows a timing diagram of a process transmitting into a web (without any complicating naks). Increasing time is towards the bottom of the figure. The transmitting process is obligated to

retransmit requested packets for at least retention heartbeat intervals after their first transmission.

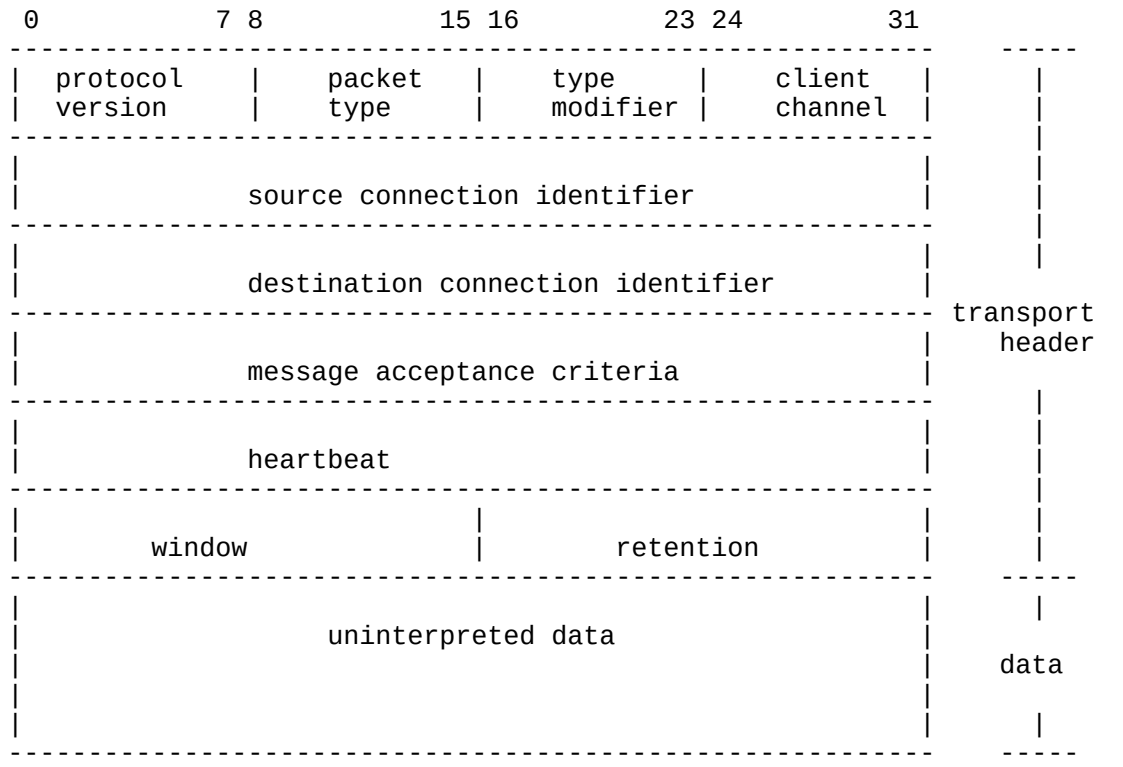

Figure 7. data packet

3.2.3. Empty packets

An empty packet is a control packet multicast into the web at regular intervals by a producer possessing a transmit token when no client data is available. Empty packets are sent to maintain synchronization and to advertise the maximum sequence number of the producer. It provides the opportunity for consuming processes to detect and request retransmission of missed data as well as identifying the owner of a transmit token.

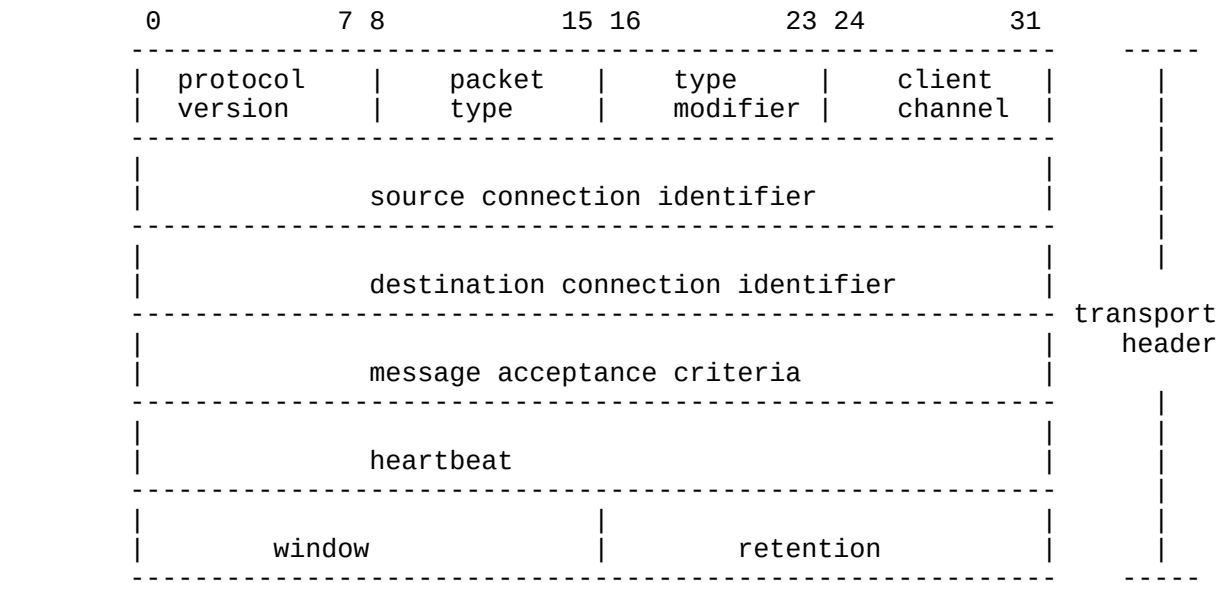

Figure 8. empty packet

There are two situations where the empty[dally] packet is used. The first is when there is insufficient data for a full packet presented by the client during a heartbeat. Partial packets should not be transmitted unless there is a client transition to be conveyed, yet something must be transmitted during a heartbeat or the master may think the process owning a transmit token has failed. Empty[dally] is used instead of a data packet until the client provides additional data to fill a packet or indicates a state transition such as an end of message or subchannel transition.

The second situation where empty[dally] is used is after the transmission of short messages. Each message should consist of multiple packets in order to enhance the possibility that consumers will observe at least one packet of a message and therefore be able to identify the producer. The transport parameter retention has approximately the correct properties for that insurance. Therefore, a message must consist of at least retention packets. If the client data does not require that many packets, empty[dally] packets must be appended. A process that has no transmittable data and is in possession of a transmit token must send an empty[cancel]. Transmissions of empty[cancel] packets pass the ownership of the transmit token back to the master. When the master observes the control packet, it will mark the referenced to message as rejected so that other consumers do not believe the message lost and attempt to recover.

During periods of no activity (i.e., after all messages have been either accepted or rejected and there are no outstanding transmit tokens) the master may enter hibernation mode by transmitting empty[hibernate] packets. In that mode the master will increase the value of the transport parameter heartbeat in order to reduce network traffic. Such packets are used to indicate that the packet's heartbeat field should not be used for resource computation by those processes that observe it.

3.2.4. Missed data

The most common method of detecting data loss will be the reception of a data or a heartbeat message that has a sequence number greater than expected from that producer. The second most common method will be a message fragment (missing the end of message) and seeing no more data or empty packets from the producer of the fragment for more than a single heartbeat. In any case the consumer process directs a negative acknowledgment (nak) to the producer of the incomplete message. The data field of the nak message contains a list of ascending sequence number pairs the consumer needs to recover the missed data.

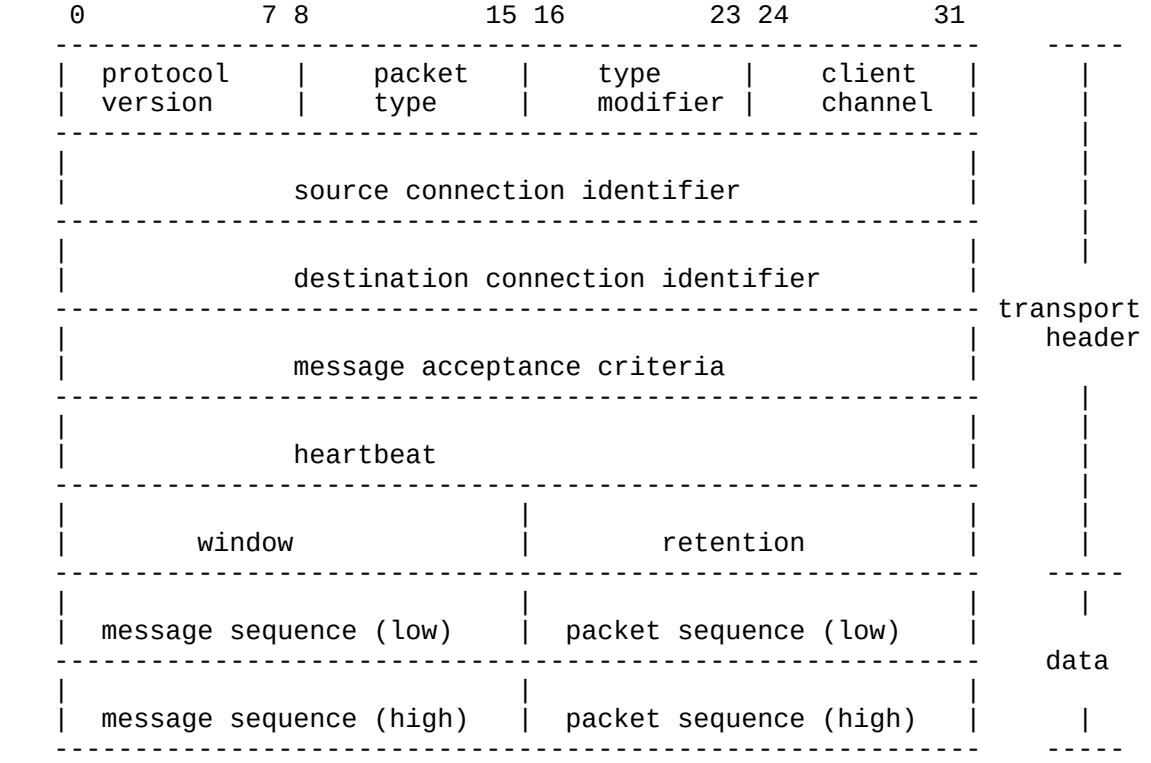

Figure 9. nak packet

3.2.5. Retrying operations

Operations must be retried in order to assure that a single packet loss does not cause transport failure. In general the right numbers to do that with exist in the transport. The proper interval between retries is the transport’s time constant or heartbeat. The proper

number of retries is retention.

Operations that are retriable (and represented by their respective message types) are join, nak, token, isMember and quit. Another application for the heartbeat and retention is when transmitting empty messages. Empty[dally] messages are transmitted any time data is not available but the data[eom] has not yet been sent. Any process not observing data or empty for more than retention heartbeat intervals will assume to have failed or partitioned away and the transport will be abandoned.

3.2.6. Retransmission

If the producer receives a nak[request] from a consumer process requesting the retransmission of a packet that is no longer available, the producer must send a nak[deny] to the source of the request. If that puts the consumer in a failed state, the consumer will initiate the withdrawal from the web. If a producer receives a nak[request] from a consumer requesting the retransmission of one or more packets, those packets will be multicast to the entire web [6]. All will contain the original client information (such as subchannel and end of message state) and message and packet sequence number. However, the retransmitted packets must contain updated protocol parameter information (heartbeat, window and retention). Retransmitted packets are subject to the same constraints regarding heartbeat and window as original transmissions. Therefore the producer’s retransmissions consume a portion of the allocation window allowing less new data to be transmitted in a single heartbeat. Retransmitted packets have priority over (i.e., should be transmitted before) new data packets.

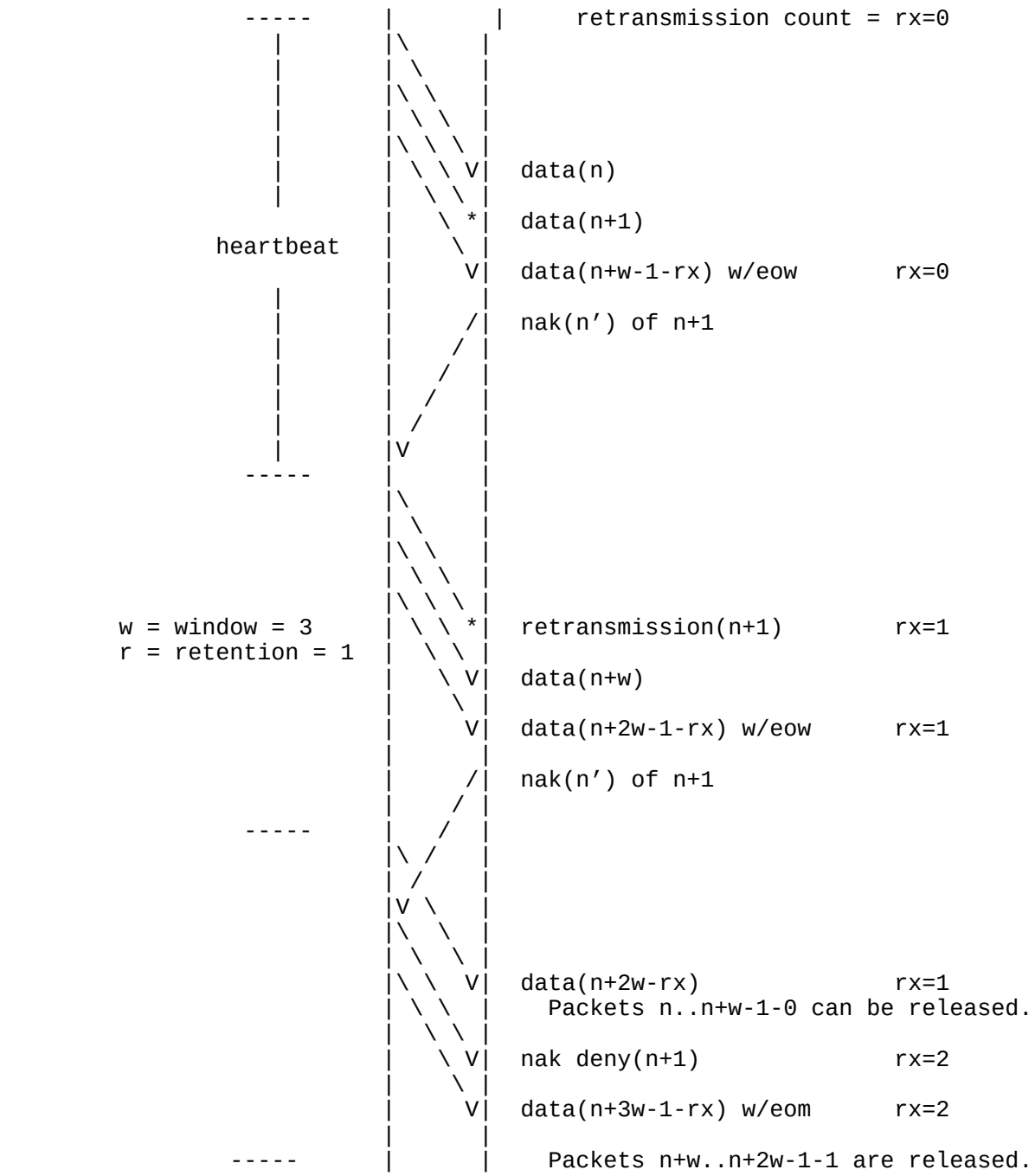

Figure 10. naks and retransmission

3.2.7. Duplicate suppression

The consumer must be prepared to ignore duplicate packets received. They will invariably be the result of the producer's retransmission in response to another consumer's nak.

3.2.8. Banishment

If at any time a process detects another in violation of the protocol it may ask the offending process to withdraw from the web by unicasting to it a quit[request] that has the target field set to the value of the offender's TSAP. Any member that exhibits a detectable and recoverable protocol violation and still responds willingly to the quit[request] will be noted as having truly correct social behavior.

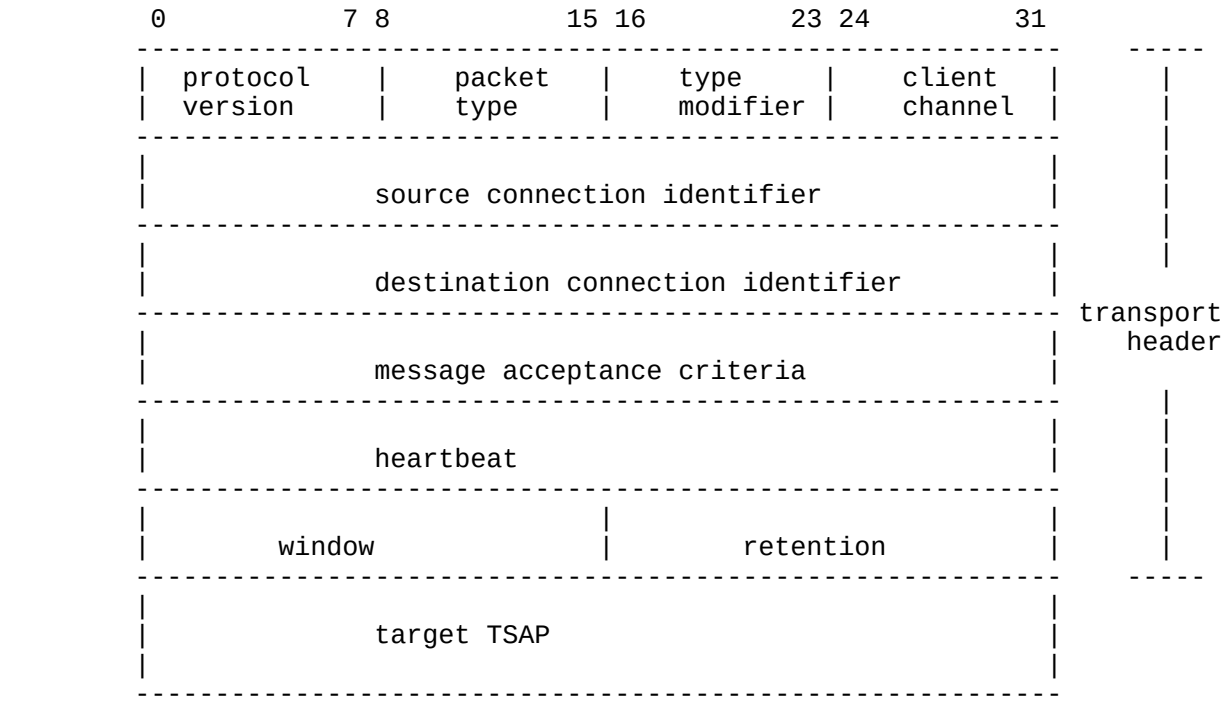

Figure 11. quit packet

3.3 Terminating the transport

Transport termination is an advisory process that may be initiated by any member of the web. No process should intentionally quit the web while it has retransmittable data buffered. Stations should make

every reasonable attempt advise the master of their intentions to withdraw, as their departure may collapse the topology of the web and eliminate the need to carry multicast messages across network boundaries.

3.3.1. Voluntary quits

Voluntary quit[requests] are unicast to the master's TSAP. When the master receives a quit from a member of the web, it responds with a quit[confirm] packet. At that time the member will be formally removed from the web. The request should be retransmitted at heartbeat intervals until the confirmation is received from the master or as many times as the web's value of retention.

3.3.2. Master quit

If the master initiates the transport termination it effects all members of the web. The master will retain all transmit tokens and refuse to assign them. Once the tokens are acquired, the master will multicast a quit[request] to the entire web. That request should be acknowledged by every active member. When the master receives no confirmations for retention transmissions, it may assume every member has terminated its transport and then may follow suit.

3.3.3. Banishment

If the master receives any message other than a join[request] from a member that it does not recognize, it should transmit a quit[request] with that process as a target. This covers cases where the consumer did not see the termination reply and retransmitted its original quit request, as well as unannounced and rejected consumers.

3.4 Transport parameters

The following section provides guidelines and rationale for selecting reasonable transport quality of service parameters. It also describes some of the reasoning behind the ranges of values presented.

3.4.1. Quality of service

Active members of the web may suggest changes in the transport's quality of service parameters during the lifetime of the transport. Producers in general adjust the transport's parameters to encourage a higher level of throughput. Since consumers are responsible for certifying reliable delivery, it is expected that they will provide the force encouraging more reliability and stability. Both are trying to optimize the quality of service. The negotiation that took place when members joined the web included the clients' desires with

regards to the worst case behavior that will be tolerated. If a member cannot maintain the negotiated lower bound, it may asked to withdraw from the web. That process will be sent a unicast message (quit[request]) indicating that it should retire. There are essentially three parameters maintained by the transport that reflect the client's quality of service requirements: heartbeat, window and retention. These three parameters can be adapted by the transport to reflect the capability of the members, the type of application being supported and the network topology. When members join the web, they suggest values for the quality of service parameters to the master. If the parameters are acceptable, the master will respond with the web's current operating values. During the lifetime of the web, it is expected that the parameters be modified by its members, though they may never result in a quality of service less than the lower bounds established by the joining procedure. Producers may try to improve performance by reducing the heartbeat interval and increasing the window size. This will have the effect of increasing the resources committed to the transport at any time. In order to keep the resources under control, the producer may also reduce the retention.

Consumers must rely on their clients to consume the data occupying the resources of the transport. To do so the consumer transport implementation must monitor the level of committed resources to insure that it does not exceed its capabilities. Since MTP is a NAK based protocol, the consumer is required to tell the producer if a change in parameters is required. The new information must be delivered to the producer(s) before the consumer's resource situation becomes critical in order to avoid missing data.

For more stable operation, consumers would try to extend the heartbeat interval and reduce the window. To a certain degree, they could also attempt to reduce the value of retention in order to reduce the amount of resources required to support the transport. However, that requires a more stringent real-time capability.

3.4.2. Selecting parameter values

The value of heartbeat is approximately the transport time constant. Assuming that the transport can be modelled as a closed loop system function, reaction to feedback into the transport should settle out in three time constants. In a transport that is constrained to a single network, the dominant cause of processing delay of the transport will most likely be page fault resolution time.

For example, using a one MIP processor on a ethernet and an industry standard disk, the worst case page fault resolution requiring two seeks (one to write out a dirty page, another to swap in the new page) and an average seek time of 40 milliseconds, page fault

resolution should be less than 80 milliseconds. Allowing for some additional overhead and scheduling delays, two times the worst case page fault resolution time would appear to be the minimum suitable transport time constant one could expect. So,

Heartbeat (minimum) = 160 - 200 milliseconds.

The transmit time for a full (ethernet) packet is approximately 1.2 milliseconds. Processing time should be less than 3 milliseconds (ignoring possible overlapped processing). Assuming disk access (with no faulting) is equivalent, and the total time per packet is the sum of the parts, or 8.4 milliseconds. Therefore, the theoretical maximum value would be approximately 17 packets per heartbeat. The transport should be capable of approximately 120 packets per second, or 19.2 packets per heartbeat.

Window (maximum) = 17 - 20 packets per heartbeat.

The (theoretical) throughput with these parameters in effect is 180 kilobytes per second.

Reducing retention may introduce instability because the consumers will have less opportunity to react to missing data. Data can be missed for a variety of reasons. If constrained to the local net the data lost due to data link corruption should be in the neighborhood of one packet in every 50,000 (bit error rate of approximately 10-9). Telephony links (between routers, for instance) exhibit similar characteristics. Several orders of magnitude more packets are lost at receiving processes, including packet switch routers, than over the physical links. The losses are usually a result of congestion and resource starvation at lower layers due to the processing of (nearly) back to back packets. The incidental packet loss of this type is virtually unavoidable. One can only require that a receiving process be capable of receiving some number of back to back packets successfully, and that number must be at least greater then the value of window. And beyond that the probability of success can be made as close to unity as required by providing the receiver the opportunity to observe the data multiple times.

The receiving process must detect packet loss. The simplest method is to notice gaps in the received message/packet sequence numbers. Such detection should be done after receiving an end of window or other state transition indication. As such, the naks cannot be transmitted, let alone received, until the following heartbeat. In order to not have any single packet loss cause transport failure, the naks should have the opportunity to be transmitted at least twice.

When the loss is detected, the nak must be transmitted and should be

received at the producing process in less than two heartbeats after the data it references was transmitted. Again, it is the detection time that dominates, not the transmission of the nak.

Retention (minimum) = 3.

The resources committed to a producing transport using the above assumptions are buffers sufficient for 80 packets of 1500 bytes each. Each buffer will be committed for 600 - 800 milliseconds.

Transports that span multiple networks have unique problems. One such problem is that if a router drops a packet, all the processes on the remote network may attempt to send a nak[request] at the same time. That is not likely to enhance the router's quality of service. Furthermore, it is obvious that any one nak[request] will suffice to prompt the producer to retransmit the desired packet. To reduce the number of nak[requests] in this situation, the following scheme might be employed.

First, extend the value of retention to a minimum value of N. Then use a randomizing function that returns a value between zero and N - 2, choose how many heartbeat intervals to dally before sending the nak[request], thus spreading out the transmissions over time. In order for the method to be meaningful, the minimum value of retention must be adjusted.

Retention (minimum) = 5 (for internet cases)

3.4.3. Caching member information

In order to reduce transport member interaction and to enhance performance, a certain amount of caching should be employed by producing members. These caches may be filled by gleaning information from reliable sources such as multicast data or, when all else fails, from responses solicited from the web's master by use of the isMember[request]. IsMember[request] requests are unicast to a member that is believed to have an accurate state of the web, at least to the degree that it can answer the question posed. The destination of such a message is usually the master. But in cases where a process (such as the master) wants to verify that a process believes itself to be valid, it can assign the target TSAP and the destination to be the same. It is assumed that every process can verify itself.

If the member receiving the isMember[request] can confirm the target's active membership status in the web, it responds with a unicast isMember[confirm]. The data field contains the credibility value of the confirmation, that is the time (in milliseconds) since the information was confirmed from a reliable source.

Caches are risky as the information stored in them can become stale. Consequently, with only a few exceptions, the entries should be aged, and when sufficiently old, discarded. Ideally they may be renewed by the same gleanable sources alluded to in the previous paragraph. If not, they are simply discarded and refilled when needed.

Web membership may be gleaned from any packet that does not have a value of unknown as the destination connection identifier. A producing transport may extract the TSAP from such packets and either create or refresh local caches. Then, if in the process of transmitting and NAK is received from one of the members whose identity is cached, no explicit request will be needed to verify the source's membership.

The explicit source of membership information is the master. Information can be requested by using the isMember message. Information gathered in that manner should be treated the same as gleaned information with respect to aging.

The aging is a function of the transport's time constant, or heartbeat, and the retention. Information about a producing member must be cached at least as long as that producer has incomplete messages. It may be cached longer. The namespace for both sequence numbers and connection identifiers is intentionally long to insure that reuse of those namespaces will not likely collide.

A. Appendix: MTP as an Internet Protocol transport

MTP is a transport layer protocol, designed to be layered on top of a number of different network layer protocols. Such a protocol must provide certain facilities that MTP expects. In particular, the underlying network level protocol must provide "ports" or "sockets" to facilitate addressing of processes within a machine, and a mechanism for multicast addressing of datagrams. These two addressing facilities are also used to formulate the NSAP for MTP on IP.

A.1 Internet Protocol multicast addressing

MTP on Internet Protocol uses the Internet Protocol multicast mechanisms defined in RFC 1112, "Host Extensions for IP Multicasting". MTP requires "Level 2" conformance described in that paper, for hosts which need to both send and receive multicast packets, both on the local net and on an internet. MTP on Internet Protocol uses the permanent host group address 224.0.1.9.

A.2 Encapsulation

The Internet Protocol does not provide a port mechanism - ports are defined at the transport level instead. In order to encapsulate MTP packet within Internet Protocol packets, a simple convergence or "bridge" protocol must be defined to run on top of Internet Protocol, which will provide MTP with the mechanism needed to deliver packets to the proper processes. We will call this protocol the "MTP/Internet Protocol Bridge Protocol", or just "Bridge". The protocol header is encapsulated the Internet Protocol data - the protocol field of the Internet Protocol packet carries the value indicating this packet is an MTP packet (92 decimal). The MTP packet itself is encapsulated in the Bridge data. Figure A.1 shows the positions of the fields within the MTP packet while table A.1 defines the contents of those fields.

A.3 Fields of the bridge protocol

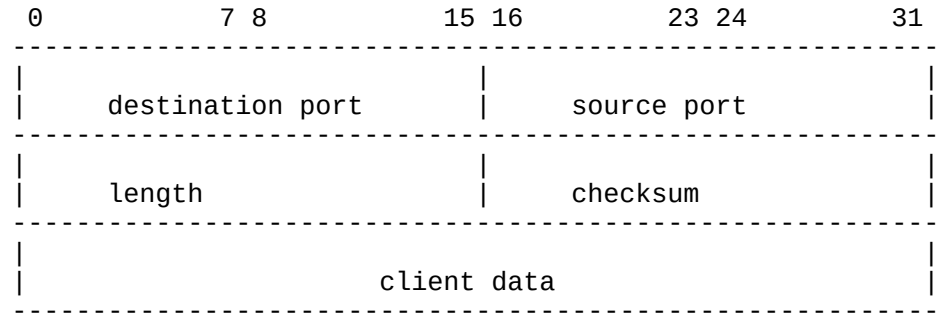

Figure A.1 MTP bridge protocol header fields

destination port The port to which the packet is destined or sinked.

source port The port from which the packet originates or is sourced.

length The length in octets of the bridged packet, including header and all data (the MTP packet). The minimum value in this field is 8, the maximum is 65535. The length does not include any padding bytes that were used to compute the checksum. Note that though this field allows for very long packets, most networks have significantly shorter maximum frame sizes - the allowable and optimal packet size must be determined by means beyond the scope of this specification.

checksum The 16 bit one’s compliment of the one’s compliment sum of the entire bridge protocol header and data, padded

with a zero octet (if necessary) to make multiple 16 bit quanities. A computed checksum of all zeros should be changed to all ones. The checksum field is optional - all zeros in the field indicate that checksums are not in use.

data The data field is the field that carries the actual transport data. A single MTP packet will be carried the data field of each bridge packet.

A.4 Relationship to other Internet Protocol Transports

The astute reader might note that the MTP/Bridge Protocol looks much like the User Datagram Protocol (UDP). UDP itself was not used because the protocol field in the Internet Protocol packet should reflect the fact that the higher level protocol of interest is MTP.

References

AFM91 Armstrong, S., A. Freier and K. Marzullo, "MTP: An Atomic Multicast Transport Protocol", Xerox Webster Research Center technical report X9100359, March 1991.

Bog83 Boggs, D., "Internet Broadcasting", Xerox PARC technical report CSL-83-3, October 1983.

BSTM79 Boggs, D., J. Shoch, E. Taft, and R. Metcalfe, "Pup: An Internetwork Architecture", IEEE Transactions on Communications, COM-28(4), pages 612-624. April 1980.

DIX82 Digital Equipment Corp., Intel Corp., Xerox Corp., "The Ethernet, a Local Area Network: Data Link and Physical Layer Specifications", September 1982.

CLZ87 Clark, D., M. Lambert, and L. Zhang, "NETBLT: A high throughput transport protocol", In Proceedings of ACM SIGCOMM ’87 Workshop, pages 353-359, 1987.

CM87 Chang J., and M. Maxemchuck. "Atomic broadcast", ACM Transactions on Computer Systems, 2(3):251-273, August 1987.

Cri88 Cristian, F., "Reaching agreement on processor group membership in synchronous distributed systems", In Proceedings of the 18th International Conference on Fault-Tolerant Computing. IEEE TOCS, 1988.

Dee89 Deering, S., "Host Extensions for IP Multicasting", RFC 1112, Stanford University, August 1989.

Fre84 Freier, A., "Compatability and interoperability", Open letter to XNS Interest Group, Xerox Systems Developement Division, December 13, 1984.

JB89 Joseph T., and K. Birman, "Reliable Broadcast Protocols", pages 294-318, ACM Press, New York, 1989.

Pos81 Postel, J., "Transmission Control Protocol - DARPA Internet Program Protocol Specification", RFC 793, DARPA, September 1981.

Xer81 Xerox Corp., "Internet Transport Protocols", Xerox System Integration Standard 028112, Stamford, Connecticut. December 1981.

Footnotes

[1] The network layer is not specified by MTP. One of the goals is to specify a transport that can be implemented with equal functionality on many network architectures.

[2] There’s only one such multicast connection identifier per web. If there are multiple processes on the same machine participating in a web, the transport must descriminate between those processes by using the connnection identifier.

[3] Determining the network service access point (NSAP) for a given instantiation of a web is not addressed by this protocol. This document may define some policy, but the actual means are left for other mechanisms.

[4] Best effort delivery is also known as highly reliable delivery. It is somewhat unique that the qualifying adjective highly weakens the definition of reliable in this context.

[5] The resource being flow controlled is packets carrying client data. Consequently, full data units provide the greatest efficiency.

[6] There seems to be an opportunity to suppress retransmissions to networks that were not represented in the set of naks received.

Security Considerations

Security issues are not discussed in this memo.

Authors' Addresses

Susan M. Armstrong
Xerox Webster Research Center
800 Phillips Rd. MS 128-27E
Webster, NY 14580

Phone: (716) 422-6437
EMail: armstrong@wrc.xerox.com

Alan O. Freier
Apple Computer, Inc.
20525 Mariani Ave. MS 3-PK
Cupertino, CA 95014

Phone: (408) 974-9196
EMail: freier@apple.com

Keith A. Marzullo
Cornell University
Department of Computer Science
Upson Hall
Ithaca, NY 14853-7501

Phone: (607) 255-9188
EMail: marzullo@cs.cornell.edu

Keith Marzullo is supported in part by the Defense Advanced Research Projects Agency (DoD) under NASA Ames grant number NAG 2-593, Contract N00140-87-C-8904. The views, opinions and findings contained in this report are those of the authors and should not be construed as an official Department of Defense position, policy, or decision.