Integrating `bytefield` and `hyperref`

Scott Pakin <scott+bf@pakin.org>

21 June 2000

Abstract

This document is a demonstration of how the `bytefield` package can integrate seamlessly with `hyperref`. The text that follows was copy-and-pasted from RFC 1301, "Multicast Transport Protocol" [AFM92].

The important thing to note is the way the fields in the protocol diagrams (drawn with `bytefield`) are hyperlinked to their descriptions (with `hyperref`). Few typesetting systems enable authors to specify hyperlinks from within a figure to the surrounding text. Fewer still do not require hyperlinks to be re-specified when the figure changes. `bytefield` + `hyperref` can do both.

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

Skipped; see [AFM92]

[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.

2.2 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.

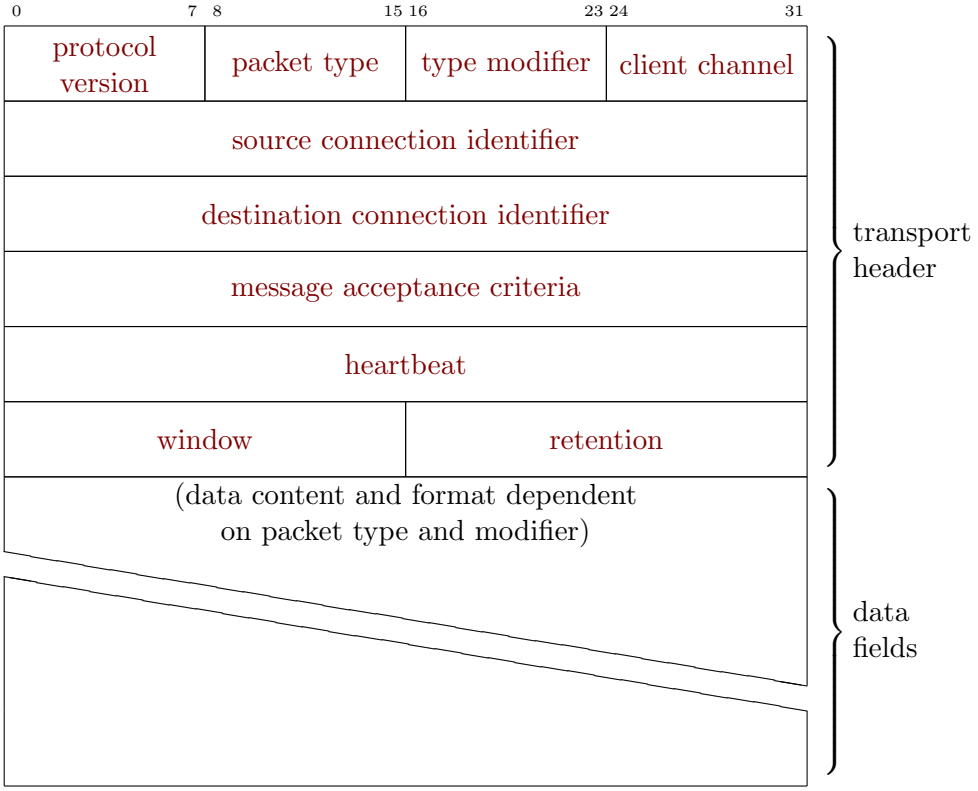

Figure 1: MTP packet format

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.

[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.

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.

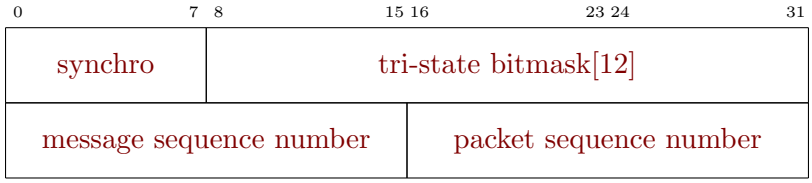

Figure 2: Message acceptance record

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.

References

[AFM91] S. Armstrong, A. Freier, and K. Marzullo. *MTP: An atomic multicast transport protocol*. Xerox Webster Research Center technical report X9100359, March 1991.

[AFM92] S. Armstrong, A. Freier, and K. Marzullo. *Multicast transport protocol*. RFC 1301, Internet Engineering Task Force, February 1992. Available from `http://www.rfc-editor.org/rfc/rfc1301.txt`.