Efficient transport and distribution of network control information in NBBS

by M. Peyravian
R. Bodner
C.-S. Chow
M. Kaplan

With the advances in fiber-based transmission systems that operate at gigabit rates and with the introduction of high-speed networks, the course of communication and computer technologies has changed forever. This change requires that new attention be focused first on the creation, then on the control of networks, which now contain high-speed links and integrate heterogeneous traffic. IBM's Networking BroadBand Services (NBBS) architecture has been designed to enable this networking revolution and, in particular, is designed for the high-speed, multimedia networks needed by emerging applications. In this paper, we present the Rapid Transport Protocol (RTP). Its simple and efficient mechanisms enable NBBS control information to be transported and distributed, taking advantage of high-speed links by eliminating as much nodal processing as possible. RTP provides point-to-point and point-to-multipoint transport services with a reliable delivery option. In addition, we present a simple and efficient mechanism for fast dissemination of time-critical network configuration and path update messages to every node in an NBBS network, making use of RTP.

High-speed networks are characterized by very high data transmission rates and high reliability. Traditional transport protocols, such as Transmission Control Protocol (TCP), may be poorly matched for the emerging environment, since they were designed for slow and unreliable networks.[1–5] In this paper, we describe Rapid Transport Protocol (RTP), which is specifically designed for fast and efficient transport and distribution of Networking BroadBand Services (NBBS) network control information.[6] RTP is used by the NBBS network control applications (such as directory services, topology services, and access agents) to transport network control information across an NBBS network. RTP transport connections are end-to-end, i.e., there is no RTP transport connection awareness in transit nodes along the path of a network connection.

RTP design began in 1988 with the objective of building a "full function" transport protocol that could exploit high-speed networks and be efficiently implemented on standard microprocessors. Greg Chesson's Xpress Transfer Protocol[4] with its "fast session setup" was taken as a starting point for the RTP design. Unlike Chesson, the IBM team did not believe it to be necessary to embed its transport protocol implementation in custom silicon. They saw that with microprocessor speeds increasing each year, their performance objectives—setup time approximately equal to network latency and

0018-8670/95/$3.00 © 1995 IBM

throughput limited only by media speed—could be met with the RTP "machine" implemented in software. Over the next few years, RTP was extended with flexible multicast and error control options to support the NBBS architecture and control protocols.

An RTP machine (an entity that supports and offers RTP services) provides to its users point-to-point and point-to-multipoint (multicast) connection-oriented services with the following features:

- RTP performs user-message segmentation and reassembly to allow user messages to flow over paths with fixed maximum packet size. When an RTP machine is requested to send a user message that would result in a packet larger than the maximum packet size, it segments the user message before sending it. The receiving RTP machine reassembles the message segments into the original user message.
- RTP provides a simple, window flow-control mechanism that prevents an RTP user from overloading its partner. An RTP user can send data only within the window granted by its partner.
- RTP has a connection-maintenance mechanism through which it can detect loss of communications with a remote partner.
- RTP allows multiple RTP transport connections to be associated with a single network connection endpoint (NCE); multiple RTP transport connections can be multiplexed onto a network connection.
- RTP provides in-order delivery of arbitrary-length user messages.
- In point-to-point transport connections, RTP provides two forms of error recovery schemes: selective repeat and go-back-n. In the selective repeat scheme, the sender only retransmits the lost data; in the go-back-n scheme, the sender retransmits the lost data and all the data that followed it.
- In point-to-multipoint transport connections, RTP enables the use of fast hardware switching mechanisms for use in point-to-multipoint multicast.

RTP is designed to be implemented over networks that are not completely reliable, and its overhead is beyond that required for a simple connectionless datagram service.[7] RTP is not a traditional transport protocol. The following RTP design principles result in high-performance implementations:

- Optimism. RTP assumes that the network and its underlying hardware components are mostly reliable, and that partner users are normally available and ready to communicate successfully. This optimism pervades the next three points.

- Fast setup for RTP transport connection establishment. RTP has no separate setup handshaking phase. The calling party simply assumes that the listening party is ready to establish an RTP transport connection. The first packet sent on an RTP transport connection may contain user data (a user message or a segment of a user message, depending on the maximum packet size).

- Data streaming and piggybacking. The calling party may continue to stream user data to the destination until status information is required. RTP assumes a listener that is able to receive this initial burst of packets. RTP makes a trade-off that favors fast connection setup and low message latency against the possibility that a few packets may reach a destination unable to process them. In most of today's communications environments, transmission is cheap, therefore dropping packets has trivial consequences compared to the cost of delaying a user's transactions.

  RTP "piggybacks" its protocol control information within packets containing user data, when convenient, to reduce the number of flows. Piggybacking control information on user data can make parsing difficult (because packets will now have variable-length headers). But RTP reduces this effect by placing the less-often-used control information in *optional segments* (as described later) and by including in the RTP packet header a payload offset field that gives the position of the user data relative to the beginning of the packet header.

- Minimized handshaking and connection dissolution. The RTP transport connection setup process and the piggybacking of control information with user data are examples of how RTP minimizes handshaking between the endpoints of the RTP transport connections it manages. This design principle also applies to connection dissolution, the process that occurs when communicating parties decide to terminate the RTP transport connection. Notification of intent to dissolve a connection is piggybacked on the packet carrying the last user message (or the last segment of the last user message).

The RTP philosophy and principles have been outlined in broad terms. The following sections delve more deeply into the detailed elements and procedures that comprise RTP.

RTP services and mechanisms

RTP provides to its users point-to-point and point-to-multipoint data transport services. It provides point-to-point services by establishing a point-to-point RTP transport connection in which one "caller" (an RTP user that assumes the calling role) is connected to one "listener" (an RTP user that assumes the listening role). Callers and listeners are distinguishable only in terms of their role in a particular connection; an RTP user may be a caller of some connections and a listener of others. In point-to-point RTP transport connections, RTP provides full-duplex transmission with two user message delivery options:

- Unreliable. If this option is selected, RTP sends the user message to its intended destination with no reliability provision. That is, if the user message is lost or not delivered to the intended recipient, RTP does not retransmit the user message nor does it inform the user that its message was not delivered correctly; however, lost user messages are reported to the receiving partner.
- Reliable. If this option is selected, RTP informs the user when the receipt of its message has been acknowledged by the intended recipient or when repeated retransmissions have failed to produce this outcome.

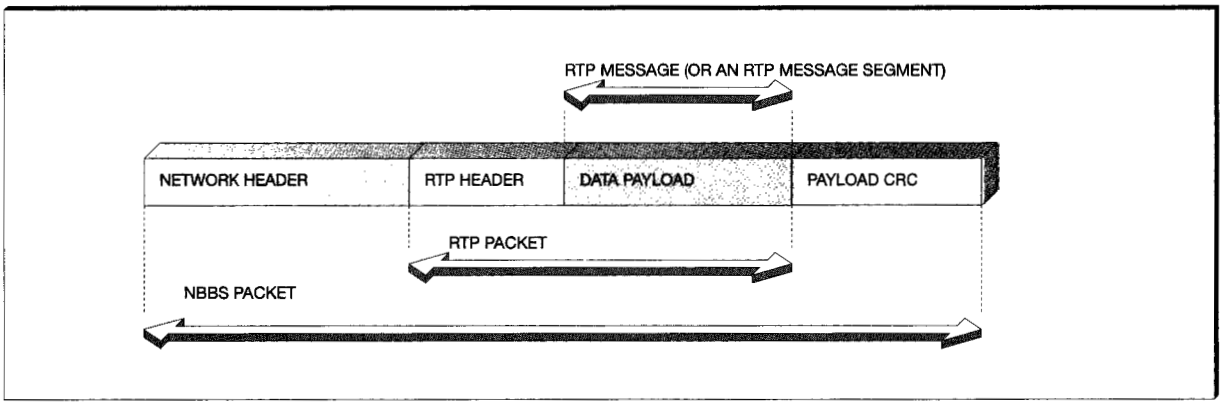
RTP provides point-to-multipoint services by establishing a multiparty RTP transport connection (MPC) in which one client application is connected to a group of server applications. The client application of an MPC initiates the MPC and has the calling role. The server applications of an MPC have the listening role. Clients and servers are distinguishable only in terms of their role in a particular MPC; a using application may be a client application of some MPCs and a server application of others. RTP provides two types of point-to-multipoint services:

- Linear multicast (LM). RTP multicasts a client-application message along a user-specified linear path to the server applications located on the path, and sends a server-application message only to the client application. RTP provides three LM service options, each of which is initiated by the client application: unreliable multicast datagram, reliable multicast datagram, and reliable multicast transaction.
- Unreliable multicast over trees (UMOT). RTP multicasts a client application message over a user-specified tree to the server applications located on the tree, and sends a server application message only to the client application. A server application may join or leave an unreliable-multicast-over-trees MPC at any time. Unreliable multicast over trees provides no reliability provision. Application messages may occasionally be lost with no indication to the sender, which can be the client application or a server application. It is up to the application to provide any required recovery.

Packet formats. Each RTP packet is comprised of a header and a data payload as shown in Figure 1. The header is of variable length and it allows for, and may include, *optional segments*. An optional segment is a special type of structure that includes control information pertaining to a transport connection. Its presence in the header depends on the type of information that needs to be conveyed from one RTP machine to its partners. When long user messages are segmented into many packets, the vast majority of the packets carry no optional segments, just a short header and the data payload. RTP supports the following optional segments:

- Status segment. This is used to convey status information from the calling to the listening partners and vice versa. It contains information such as acknowledgments for already received data and window-allocation parameters.
- Connection setup segment. This segment is used in the RTP transport connection establishment process. The calling partner sends this segment in a packet to convey the setup information to its partners. This segment contains information such as the source identifier.
- Return path information segment. This segment is used in point-to-point RTP transport connections to pass the return path information from the local partner to the remote partner.
- Linear multicast segment. This segment is used in a linear multicast RTP transport connection to carry linear multicast related information (such as service option type) from the client to servers.
- Connection fault segment. When an error occurs, this segment is used to carry sense data (identifying the error) from the partner that detected the error to the other partners.

Figure 1 RTP and NBBS packets



The data payload is of variable length and it contains a message (or a message segment) and optional padding. Cyclic redundancy checking (CRC) protects both the header and the data payload.

The base header contains a byte-sequence-number field, which identifies the first user-message byte of each packet. As long as no packets containing user data (or an end-of-message character) are lost by the underlying network, the receiver simply checks that the value in the byte sequence number field equals the sum of the byte sequence number of the previous packet and the number of data payload bytes (plus one if an end-of-message character is present) in the previous packet.

Since RTP checks the byte sequence number of each received packet, if a packet arrives with a higher than expected byte sequence number, it can immediately deduce that packets have been lost. This process of deduction of packet loss is referred to as *gap detection*. This mechanism is used for error recovery, as described later.

A sequence number is assigned to each byte of the data payload, rather than to the packet containing the data payload, because of the variable-length RTP header. If sequence numbers are assigned to packets, then a retransmission may require a longer header than that of the original transmission (because additional optional segments may have to be included in the retransmission). When packet size is limited, the packet to be retransmitted may not have enough room for additional optional segments.

Transport connections multiplexing. As part of the transport connection setup process, the calling RTP machine chooses and assigns a 4-byte transport connection identifier (TCID) to the transport connection. This TCID, along with a globally unique connection qualifier associated with the calling RTP machine (which may be a network address or an International Organization for Standardization [ISO] object identifier), is included on all packets and associates the packet with the RTP transport connection. When sending packets, the listening partner always puts the TCID value received from the calling partner in each packet it generates. Since these packets will contain the TCID chosen by the calling partner, they do not carry a connection qualifier.

RTP allows multiple RTP transport connections to be associated with a single NCE; i.e., multiple RTP transport connections can be multiplexed onto the same network connection. This is possible because an RTP machine associated with an NCE assigns a different TCID to each RTP transport connection, and because packets belonging to different RTP transport connections can be identified based on their TCIDs. Thus, an RTP machine associated with an NCE has the capability to multiplex and demultiplex RTP transport connections onto and off of network connections using the TCIDs.

Message segmentation. The network packets must be large enough to contain the variable-length RTP headers. When the network packets are not large enough to hold both the RTP header and message, RTP segments the message into pieces to fit into

the network packets at the sending side and reassembles the message segments into the original message at the receiving side. This is referred to as *message segmentation*. RTP performs message segmentation, reassembly, and sequence checking in a straightforward fashion, with start-of-message and end-of-message bits in the RTP header. When no message segmentation is required, a packet containing a message has both the start-of-message and the end-of-message bits set. When message segmentation is required, the first packet contains the first segment of the message; it has the start-of-message bit set but does not have the end-of-message bit set. The last packet contains the last segment of the message; it does not have the start-of-message bit set but does have the end-of-message bit set. Any middle packets (each containing a segment of the message) do not have either the start-of-message or the end-of-message bit set.

Window flow control mechanism. RTP maintains a *receive window* allocation, which is communicated end-to-end between partners via the allocation-sequence-number (ASEQ) field in the status segment. The partner sending a status segment has allocated a receive window and agrees to receive user message bytes with byte sequence numbers starting at received sequence number (RSEQ) and ending at ASEQ − 1. The sender of the status segment will not accept any user message bytes outside this allocated receive window; it will simply discard any such data. The receive window allocated by one partner is the *send window* of the other partner. RTP stops sending user-message bytes at byte sequence number ASEQ − 1, that is, when its send window is shut.

During the RTP transport connection setup, the calling partner assumes some initial send window (by assuming some initial value for ASEQ). This allows the calling partner to send a burst of packets containing user-message bytes without waiting for its partner to actually communicate its receive window allocation. This initial ASEQ can be an implementation-default value.

Timers. There are five timers required for the operation of RTP at each end of an RTP transport connection: a short-request (SHORT_REQ) timer, a short-response (SHORT_RSP) timer, a gap-received (GAP_REC) timer, a dally timer, and a connection-inactivity timer. There is also a maximum-retry (MAX_RETRY) count required at each end of an RTP transport connection.

The SHORT_REQ timer is for error recovery, and it provides a basic retransmission mechanism. It is used when waiting for status information from a remote partner. When an RTP machine wants to know the status of its partner, it sends a packet with the status-request bit set, indicating that it wants its partner to send a status segment, and starts its SHORT_REQ timer. When an RTP machine receives a packet with the status-request bit set, it responds by sending a packet that contains a status segment. The SHORT_REQ timer is stopped when a current status segment is received. If the SHORT_REQ timer expires, the RTP machine sends another packet with the status-request bit set.

The number of consecutive times that the SHORT_REQ timer can expire at each RTP machine is governed by MAX_RETRY. An RTP machine increases its retry count by one each time the SHORT_REQ timer expires. An RTP machine resets its retry count to zero if it receives a current status segment from its partner. When an RTP machine's retry count is greater than MAX_RETRY, it assumes that its partner has failed or has become unreachable. In this case, it reports the failure to the RTP user and disconnects.

The SHORT_RSP timer is for piggyback optimization; it specifies how long an RTP machine can wait before sending a status segment in response to one or more status requests from its partner. When an RTP machine receives a packet with the status-request bit set, it starts the SHORT_RSP timer unless it is already running. When the SHORT_RSP timer expires, it sends a packet containing a status segment to its partner and stops the SHORT_RSP timer.

The SHORT_RSP timer has two benefits. First, an RTP machine need not send a separate status segment for each received packet with the status-request bit set. Second, if user data are not yet available and a partner has requested status information, the RTP machine can wait a short time. If user data become available during that time, the status segment can be included in the packet containing the user data.

When an RTP user wants to disconnect, if the RTP machine has received reliable messages from its partner, it does not terminate the RTP transport connection immediately. Instead, it starts the dally

timer and will "dally" (wait) for a while. Dallying ensures that the remote partner receives acknowledgments for all reliable data payloads that it has sent. During the dallying period, any data payload received from the remote partner is simply discarded. The data payload may be a redundant retransmission—perhaps because a packet containing a status segment (acknowledgment) was lost. The data payload may be new (probably due to an RTP user error); or perhaps the RTP users did not "handshake" properly (perhaps an RTP user did not know that its partner has requested to disconnect).

The connection-inactivity timer provides a connection maintenance mechanism through which a failure of the remote partner or loss of communications with the remote partner can be detected. The connection-inactivity timer is started (or restarted if it is already running) each time a packet is received from the remote partner.

The GAP_REC timer is used by the receiving RTP machine (implementing the selective repeat error recovery scheme) to periodically notify the remote sending RTP machine that lost reliable message bytes have not been recovered. This timer is used to speed up the error recovery process. When an RTP machine detects a new gap in the reliable data stream, it sends to its partner a packet containing a status segment with the gap-detected-by-the-receiver (GAPDETR) bit set and starts or restarts the GAP_REC timer. If the GAP_REC timer expires, another packet containing a status segment with the GAPDETR bit set is sent and the GAP_REC timer is restarted. The GAP_REC timer is stopped when all missing reliable message bytes (including the ones in earlier gaps) are received.

Point-to-point connections

RTP connections are initiated by RTP applications. The application notifies its RTP machine of potential connections and for each one the RTP machine constructs a unique "connection context."

An RTP point-to-point transport connection begins life with one RTP user assuming the calling role and another remote RTP user assuming the listening role. How it is decided which partner is calling and which partner is listening is outside the scope of RTP.

The RTP approach to connection setup is very different from the traditional handshaking protocols of Open Systems Interconnection (OSI) transport, Systems Network Architecture (SNA), and TCP. The RTP approach is very similar to and borrows from Xpress Transfer Protocol (XTP)[4] and Delta-t.[1] Optimism is combined with piggybacking to minimize the latency of the first bytes of user data to follow on an RTP connection. Another way to think about the RTP setup protocol is to view it as an overlapping of a more traditional session establishment handshake with data transport.

Upon the request of the calling RTP user for transmission of the first message, the calling RTP machine constructs a special type of packet called a *setup packet* and sends it to the listening RTP machine. Along with the usual RTP header and user data, the setup packet contains a connection-setup segment that carries information, such as the source identifier. The setup packet may also include other optional segments.

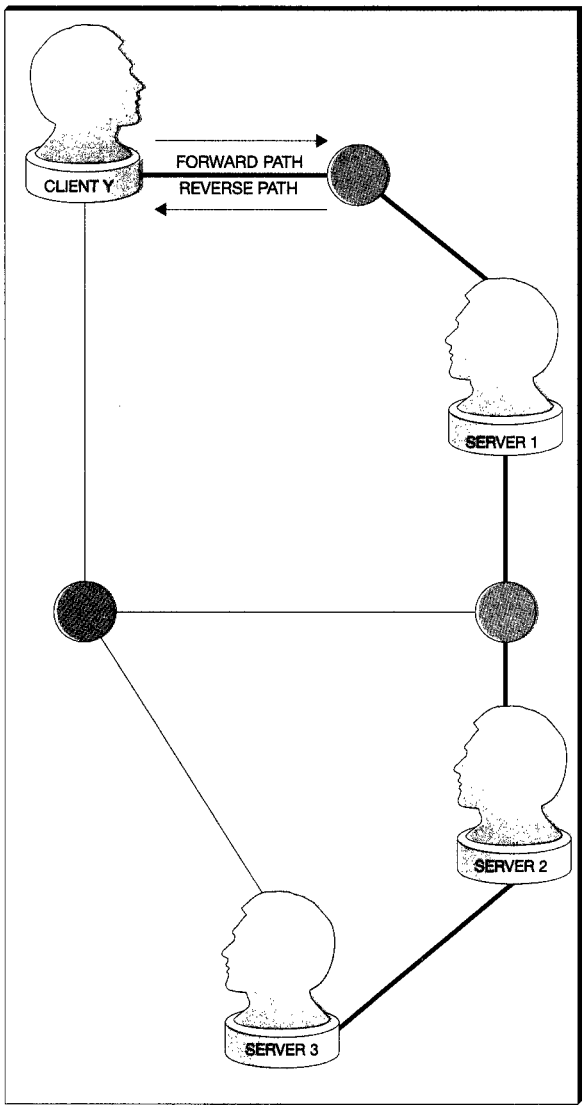
The listening RTP machine treats setup packets in a special way. If the setup information does not duplicate previously received information and there is a matching connection context, the listening RTP machine enters a "connected state." The calling RTP machine does not enter the connected state until it receives a packet indicating that the listening partner has received the setup packet. Once both the RTP machines are in the connected state, a point-to-point transport connection exists and packets may flow in either direction.

In point-to-point connections, RTP offers two options for reliable message delivery: selective repeat and go-back-n. In the selective repeat option, RTP only retransmits the lost user data bytes; however, in the go-back-n option, RTP retransmits all the user data bytes following the last acknowledged byte without an earlier gap. The procedures that RTP uses to support these two schemes are described in Appendix A.

Point-to-multipoint connections

RTP provides point-to-multipoint (multicast) services by establishing a multiparty RTP transport connection (MPC) in which one client RTP machine is connected to a group of server RTP machines. The client RTP machine of an MPC initiates the MPC and has the calling role. The server RTP machines of an MPC have the listening role. The client RTP machine multicasts packets to the server RTP machines, but a server RTP machine may send pack-

Figure 2 Typical configuration for a linear multicast MPC



ets only to the client RTP machine on the same MPC. When replies (acknowledgments or messages) from the server RTP machines are expected, packets sent by the client RTP machine have the reverse path accumulation function activated (each intermediate node determines its label for the return path and includes it in the packet). Then, packets from the server RTP machines are sent point-to-point on the same MPC to the client RTP machine with the automatic network routing (ANR) transfer mode using the accumulated reverse ANR labels in the received packets. RTP multicast services are used by the NBBS network control applications (directory services, network connection services, and topology services).

There is a using application associated with each of the RTP machines participating in an MPC; the application using the client RTP machine is the client application, and the applications using the server RTP machines are server applications. Clients and servers are distinguishable only in terms of their role in a particular MPC; a using application may be a client application of some MPCs and a server application of others. RTP currently provides two types of multicast services, linear multicast (LM) and unreliable multicast over trees (UMOT), which are described in the following sections.

Linear multicast. RTP provides LM services by establishing a multiparty RTP transport connection (MPC) in which there is a client RTP and a group of server RTPs. The RTP machine in the origin node is the client RTP. The RTP machines in the transit nodes and in the destination node are server RTPs. Normally, a client RTP multicasts packets to all the server RTPs, and a server RTP sends packets only to the client RTP. At certain times, communication may take place between the client RTP and a subset of the server RTPs. For example, this occurs when a reliable message is retransmitted only to server RTPs that have not yet acknowledged the message.

Figure 2 illustrates a typical configuration for the client and server RTPs in a linear multicast MPC. The forward and reverse paths used by the client and server RTPs to send packets are indicated.

RTP provides three LM service options, each of which is initiated by the client application:

- Unreliable multicast datagram (UMD). This service option provides delivery of a single message from the client RTP to user-specified server RTPs with no reliability provision.
- Reliable multicast datagram (RMD). This service option provides reliable delivery of a single message from the client RTP to user-specified server RTPs.
- Reliable multicast transaction (RMT). This service option provides reliable delivery of a single message from the client RTP to user-specified

server RTPs and delivery of one or more messages (only the last of which is reliable) from each server RTP to the client RTP.

Each of these service options is implemented using a short-lived MPC. The client application may interleave UMDs, RMDs, and RMTs in any order desired by the particular application. The procedures and mechanisms that RTP uses to provide these three service options are described in Appendix B.

The client RTP sends packets to the server RTPs with the ANR transfer mode. With this mode the entire path of the packet is identified at the origin of the connection and placed in the network header of the packet as an ANR field. When a node receives the packet, it extracts the ANR label that identifies the link over which the packet was received from the ANR field and then forwards the packet to the next node in the path. The next node will then repeat the process.

When multicasting packets to the server RTPs, the client RTP utilizes the selective copy function of the ANR transfer mode. The selective copy function allows only certain nodes along a path to copy the packet, eliminating unnecessary processing of packets. The client RTP, located in the origin node, multicasts a packet by including an ANR field in the network header and sending the packet along the path specified by the ANR field to the endpoint server RTP located in the destination node.

Sometimes it is not necessary for the client RTP to send a packet to all the server RTPs for an MPC; for example, an acknowledgment can be sent only to the server RTPs from which a reliable message has been received, and a reliable multicast message needs to be retransmitted only to the server RTPs that have not acknowledged the multicast message. Delivery to the server RTP in a transit node is specified with the selective copy bit of the ANR label for the outgoing link of the node.

Unreliable multicast over trees. The client RTP initiates an unreliable multicast over trees (UMOT) MPC and multicasts packets to all the server RTPs located on the tree. On the same MPC, a server RTP may send packets only to the client RTP. When a packet is sent over a tree, the packet will be forwarded to all members of the tree (all the server RTPs located on the tree). Packets sent by the client RTP use a label-swap transfer mode (which swaps an inbound label for an outbound label via hardware) and have the reverse path accumulation function activated. Packets from a server RTP are sent on the same MPC to the client RTP with the ANR transfer mode, using accumulated reverse ANR labels (which were collected by the reverse path accumulation function).

RTP provides no reliability when an application requests the UMOT services. Application messages may occasionally be lost with no indication to the sender (which might be the client application or a server application). UMOT provides delivery of one or more unreliable messages from the client RTP to all the server RTPs, and delivery of zero or more unreliable messages from each server RTP to the client RTP.

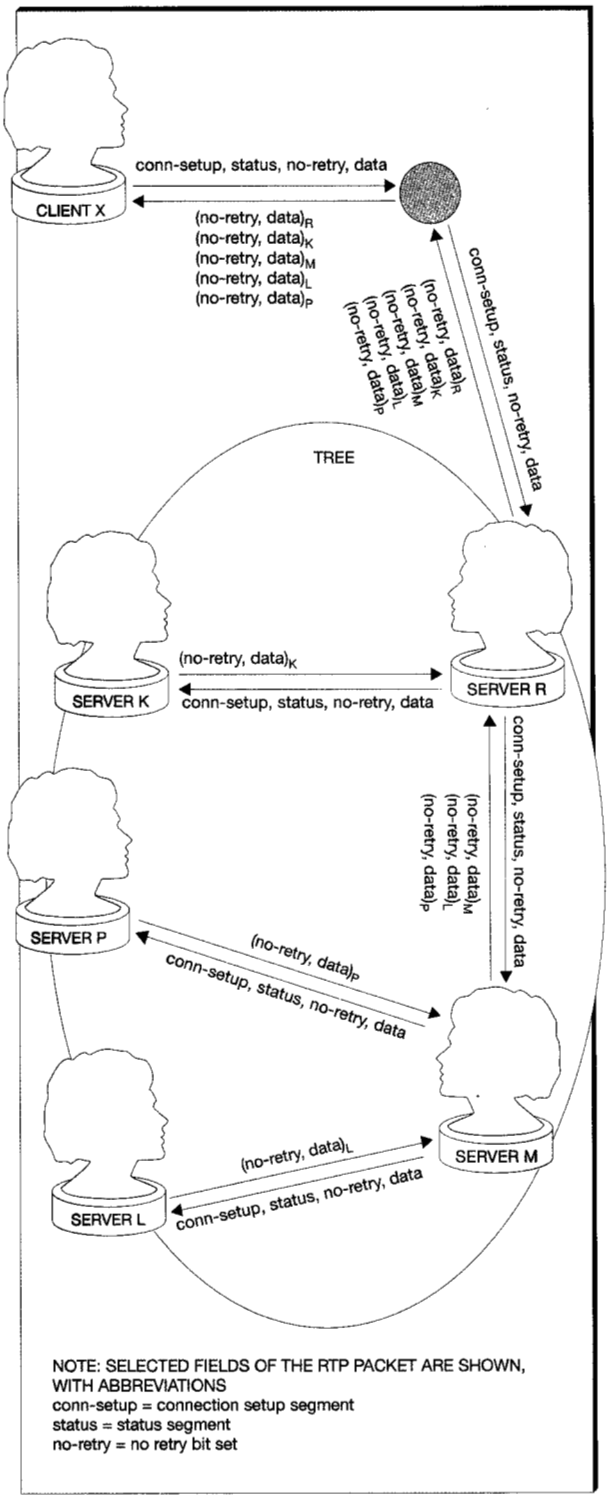
Unreliable multicast over trees allows a server RTP to join or to leave an ongoing MPC at any time. That is, a server RTP need not participate in an MPC during the entire life of the MPC. This allows server RTPs that have failed and recovered to rejoin the MPC.

Figure 3 illustrates an example of client and server RTP transmissions in an UMOT MPC. The procedures and mechanisms that RTP uses to provide UMOT MPC are described in Appendix C.

Reliability for the UMOT service can be provided by an application that uses this service. The next section describes a topology-based reliability mechanism, provided by NBBS topology services, which makes use of the UMOT service to multicast time-critical configuration information to the entire NBBS network.

An example of reliability added by an application to the UMOT service. The objective of NBBS topology services is to maintain a consistent view of the network topology in each node in the network. A consistent view is required for finding key services, maintaining current network state information, and computing good paths through the network. The main requirement on topology services is to ensure that all nodes in the connected portion of the network eventually acquire the same picture of the network. Since this information is critical, it must be distributed quickly, efficiently, and reliably. Topology services enable the information to be distributed quickly and efficiently via an underlying structure termed the *control point* (CP) *spanning tree*, which dynamically identifies the nodes that

Figure 3 Client and server RTP transmissions in an UMOT MPC



are part of the MPC comprising the entire NBBS network, and which enables packet transfer in the hardware. Topology services also provide an application-based reliability mechanism that makes use of the RTP UMOT service. This section describes the creation and maintenance of the CP spanning tree and the application-based reliability provided by topology services.

Topology services create and maintain a CP spanning tree for distribution of time-critical, vital configuration information. In NBBS, the CP spanning tree is defined to be a graph comprised of one or more nodes and zero or more edges; thus a CP spanning tree could be a single node. An edge is a link (transmission medium) that is part of the CP spanning tree. Each link is represented as a pair of *unidirectional links*, where one unidirectional link enters the node and one unidirectional link emanates from the node. Each unidirectional link has a label associated with it to enable fast packet transfer in the hardware. This label is called an *ANR label.*

Each node on the CP spanning tree maintains a parent-child relationship with its adjacent nodes. Each node has one parent and zero or more children, except for the root, which has zero or more children but no parent. Every node in the CP spanning tree is capable of being the root. The property of being the root is termed *rootship*.

The root coordinates the construction and maintenance of the CP spanning tree and initiates the process of joining two CP spanning trees together. When a node becomes the root of a CP spanning tree, it determines where the CP spanning tree join will take place.

If the node (where the CP spanning tree join should take place) is not the current root of the CP spanning tree, the rootship is transferred hop-by-hop to the node that owns that unidirectional link via a move root message. Each node, upon receipt of a move root message, determines which node owns the highest-weight potential unidirectional edge.[8] If the node determines that it does not own the highest-weight potential unidirectional edge, it calculates a path to the node that does and sends a move root message to the next node along the path. If the node determines that it is the root where the CP spanning tree join should take place, it begins the process of joining the two CP spanning trees.

Each root compares its node identifier with that of the adjacent node to which the potential unidirectional edge is attached to determine if it is the lower- or higher-named node. The comparison is done using the extended binary coded decimal interchange code (EBCDIC) collating sequence. (For example, the EBCDIC value for "A" is less than the EBCDIC value for "9," therefore "9" is higher than "A.") Then the node takes the following actions:

1. If the root is attached to the link where the join should take place and is the lower-named node, it sends a combine request message to the higher-named node, using the reliable point-to-point RTP service.

2. Upon receipt of the combine request message, the higher-named node does one of the following:

   - If the CP spanning tree of the higher-named node is in the process of joining with another CP spanning tree or if the higher-named node has not yet received a move root message, it does nothing after receiving the combine request message until one of the following events happens:
     a. The join with the other spanning tree is completed. Then the higher-named node becomes the root of its CP spanning tree.
     b. It receives a combine reply demanded message. This notifies the higher-named node that the lower-named node wants to withdraw its joint offer because it has received information about a new highest-weight potential unidirectional edge. The higher-named node then does one of the following:
        i. If it has already agreed to the join offer (it has already sent a combine grant message to the lower-named node) or it is ready to agree, it discards the combine reply demanded message, sends the combine grant message (if not already sent), and the join continues normally.
        ii. If it is ready to accept the lower-named node's join offer (because it is in the process of joining with another CP spanning tree), it sends a combine release message to the lower-named node, and the CP spanning tree join proceeds no further.

   - If the higher-named node is not already in the process of joining with another CP spanning tree and has received a move root message, it accepts the join offer (the combine request) by sending a combine grant message to the lower-named root, using the reliable point-to-point RTP service.

3. Each node creates and sends its configuration information to the other node involved in the join. The lower-named node does this when it receives the combine grant message and the higher-named node when it sends the combine grant message. Upon receipt of this configuration information, each node will multicast any new configuration information to its own CP spanning tree and any inconsistent configuration information to both CP spanning trees, so that all nodes will correct the information.

4. Each node marks the hardware of the unidirectional link emanating from it with hardware addresses to facilitate fast packet transfer: topology services in the lower-named node do this after processing configuration information from the higher-named node, then send a combine done message to the higher-named node; topology services in the higher-named node do this after processing configuration information from the lower-named node and upon receipt of the combine done message from the lower-named node.

5. When the respective nodes are finished marking the unidirectional links with the hardware addresses, the lower-named node (that sent the combine request) makes its parent the higher-named node and the higher-named node (that received the combine request) becomes the root of the newly combined CP spanning tree.

The combined CP spanning tree is the new vehicle for transfer of configuration and utilization information. It dynamically identifies the nodes that are part of the MPC (for use by the UMOT RTP service) and enables (via the hardware-marked address of the link) fast packet transfer. The CP spanning tree, along with the UMOT RTP service distribution, enables NBBS to issue topological information with the distribution of only $n - 1$ messages each time a multicast occurs, where n is the number of nodes in the network (in the CP spanning tree). The UMOT RTP service and the creation and maintenance of the CP spanning tree are more efficient than a flooding-type algorithm (such as the ARPANET algo-

rithm[8]) for the distribution of topological information.

Since configuration information is vital to the network operation, topology services provide a reliable guarantee for any configuration information issued over the CP spanning tree using the UMOT RTP service, as described below.

Each time the topology services application multicasts configuration information using the UMOT RTP service, it uniquely identifies the multicast by a reliable multicast correlator and a node identifier. The reliable multicast correlator is an integer that the application increases by one each time it uses the UMOT service to multicast configuration information. The node identifier identifies the node that issues the multicast.

Immediately after using the UMOT RTP service to multicast the configuration information, topology services use the unreliable point-to-point RTP service to send a reliable multicast (RM) acknowledgement (ack) message to each of its CP spanning tree neighbors; the RM ack message also contains the reliable multicast correlator and node identifier identifying the multicast event. Topology services then set a timer called the RM-ack timer, which indicates the length of time the node will wait for RM ack messages from its neighbors. The RM ack messages are used as a "handshake" between the neighbors, indicating that the node has received (or sent, in the case of the originator) the configuration information identified by the reliable multicast correlator and node identifier. Each node that receives the configuration information also sends an RM ack message to each of its neighbors, sets its RM-ack timer, and performs the following events.

When the node sets the RM-ack timer, it also creates an RM-acks-expected list that tracks which neighbors send RM ack messages for the specified multicast event. When creating the RM-acks-expected list, the node determines whether an RM-acks-received list exists for the multicast event (note that this is not necessary for the node that originated the multicast of the configuration information). When a node receives an RM ack message from a neighboring node identifying a multicast event for which no RM-acks-expected list exists, the node creates an RM-acks-received list and places the ANR label, over which it received the unexpected RM ack message, in this list. This scenario occurs whenever the configuration information for a specified multicast event is lost (e.g., due to buffer overflow, the packet containing configuration information passes through the hardware but a copy of the packet is not made). When the node receives additional RM ack messages from other CP spanning tree neighbors for the same multicast event, the node puts these ANR labels, over which the RM ack messages flowed, into the RM-acks-received list, as long as the configuration information for the corresponding multicast event has not been received. Once the configuration information is received by the node, topology services create an RM-acks-expected list.

New configuration information will flow only to the node's own CP spanning tree, making topology services more efficient.

The ANR labels emanating from all CP spanning tree neighbors and entering the node are candidates for the RM-acks-expected list. If an RM-acks-received list exists, it is used to trim entries from the RM-acks-expected list; the ANR labels over which RM ack messages have already been received are not placed in the RM-acks-expected list. Neither does the node put ANR labels of all neighboring nodes in the RM-acks-expected list when new configuration information is received during a CP spanning tree join—the ANR label over which the join is taking place is not placed in the RM-acks-expected list. This is desirable because the new configuration information will flow only to the node's own CP spanning tree, making topology services more efficient (the messages do not flow through both CP spanning trees since it is not necessary).

The RM-acks-received list is then discarded since it is no longer needed (the configuration information has been received), RM ack messages and reliable point-to-point configuration information (corresponding to the multicast event) are tracked by removing the ANR labels, over which they are received, from the RM-acks-expected list corresponding to the multicast event.

When the RM-ack timer expires (if the RM-acks-expected list does not empty prior to the RM-ack timer expiring), the node sends another copy of the configuration information to each neighbor that did not send an RM ack message—but this time using the reliable point-to-point RTP service. (Sending the configuration information reliably via RTP ensures that it gets to the neighbor.) The node then discards the RM-acks-expected list, since the configuration information arrived at each of the neighboring nodes—by the UMOT RTP service or by the reliable point-to-point RTP service.

The node then sets the RM-event timer, which is used to eliminate unnecessary processing of late RM ack messages or configuration information for the specified multicast event. Late responses could be caused by transient loops in the CP spanning tree (which may occur when the CP spanning tree configuration changes faster than the information about these changes is delivered) or by out-of-order delivery of messages. Once the RM-event timer expires, the multicast event information for the specified multicast event is discarded (enough time has transpired that the probability of either event has been greatly reduced).

If the configuration information is not received via multicast but only by a reliable point-to-point message, the node forwards the configuration information to each of its CP spanning tree neighbors (except the one from which it received the information) using the reliable point-to-point RTP service. Immediately after sending this information to its neighbors, the node sets the RM-event timer. (The RM-acks-expected list and RM-acks-received list are not created nor is the RM-ack timer set, in this scenario.)

With this topology-based reliability, topology services guarantee that the multicast configuration information reaches every node in the network.

Conclusion

We introduced RTP, a high-performance protocol that provides point-to-point and point-to-multipoint transport services with a reliable delivery option. This protocol enables NBBS network control information to be transported and distributed in a manner that takes advantage of high-speed links by eliminating as much nodal processing as possible. We also presented a simple and efficient mechanism, using the UMOT service of RTP, for fast and efficient dissemination of time-critical network configuration and path update messages to every node in an NBBS network. For the distribution of the network configuration information, we presented an example of an application-based reliability mechanism for these multicast messages.

In retrospect, what distinguishes RTP from XTP and the other "light-weight" protocols proposed during the late 1980s and early 1990s, other than technical details, is that the RTP design evolved to perfectly support the rest of the IBM NBBS architecture and that RTP has been successfully implemented and deployed in several IBM products. RTP is not a theory. It became more than a research project. The several implementations of RTP prove that "fast, optimistic session setup" works and that sensible packet formats can be handled efficiently with straightforward code running on off-the-shelf microprocessors.

Acknowledgments

The authors would like to express appreciation and acknowledge the contributions of Gary Dudley for his work on RTP development. Appreciation is also expressed to Marcia Peters, Phillip Chimento, and Shay Kutten for their work during the early development of topology services, and to Theodore Tedijanto and Jean-Paul Chobert for their topology services contributions.

Appendix A: Reliable delivery by retransmission

This appendix describes the retransmission procedures that RTP uses to detect and recover lost packets. One partner informs the other partner about its status by including a status segment in a packet sent to the partner. The status segment contains acknowledgments for already received reliable message bytes. One partner requests a status segment from its partner by setting the status-request bit in the RTP header. The loss of a transmitted packet containing message bytes is detected at the receiver through a gap in the sequence number.

An RTP user can suppress error recovery by sending messages unreliably. The sending RTP machine sets the no-retry bit in the containing packets and does not retransmit these messages. The receiving RTP machine does not report missing message bytes back to the sender from a packet with the

no-retry bit set. If status is required, the receiving machine sends the status segment back to the sending machine with the byte-sequence-number field updated to reflect the reception of the most recently received message bytes. It discards any unreliable message that has a gap in it, and reports the missing (or discarded) unreliable message to the receiving RTP user.

When gaps are detected in reliable messages, RTP uses the procedures described next to recover missing message bytes.

Selective repeat. When a receiving partner, B, detects a new gap, it buffers the message bytes that follow the gap. Then, partner B builds a packet containing a status segment with the gap-detected-by-the-receiver (GAPDETR) bit set. The status segment acknowledges the most recently received message bytes without an earlier gap. It also acknowledges any safely arrived and buffered message bytes following the oldest existing gap so that the sending partner, A, will know that it does not need to retransmit those spans of message bytes. For this purpose the status segment carries zero, one, or more acknowledged byte span pairs (ABSP). Each ABSP represents a sequence of message bytes that are being held in partner B's buffers pending the arrival of the message bytes expected based on the gaps. Partner B sends this packet to partner A and starts (or restarts) the GAP_REC timer. The GAP_REC timer is not stopped unless all the lost message bytes (including the ones in earlier gaps) are recovered. As long as the lost message bytes are not recovered, any subsequent status segment sent (in a packet) by partner B will have the GAPDETR bit set, reporting all the existing gaps. If the GAP_REC timer expires, partner B restarts it and sends another packet, containing a status segment with the GAPDETR bit set, to partner A.

In order to avoid retransmitting a message byte if it is not lost, partner A will retransmit a message byte (reported as being in a gap) only if a transmission following it in time (regardless of its byte sequence number) has been acknowledged. For this purpose partner A maintains a *chained buffer* in which it retains the message bytes that have been transmitted but not yet acknowledged. The message bytes in the chained buffer are chained in the order of the most recent transmission. Thus, when a message byte is transmitted (or retransmitted) it is placed at the end of the chained buffer. Whenever partner A receives a status segment (current or old) from partner B, it removes each acknowledged message byte from the chained buffer. In addition, partner A records the point in the chained buffer where the most recently transmitted message byte is acknowledged. Each message byte in the chained buffer that was last transmitted, prior to this most recently acknowledged message byte, is referred to as "eligible-for-retransmission."

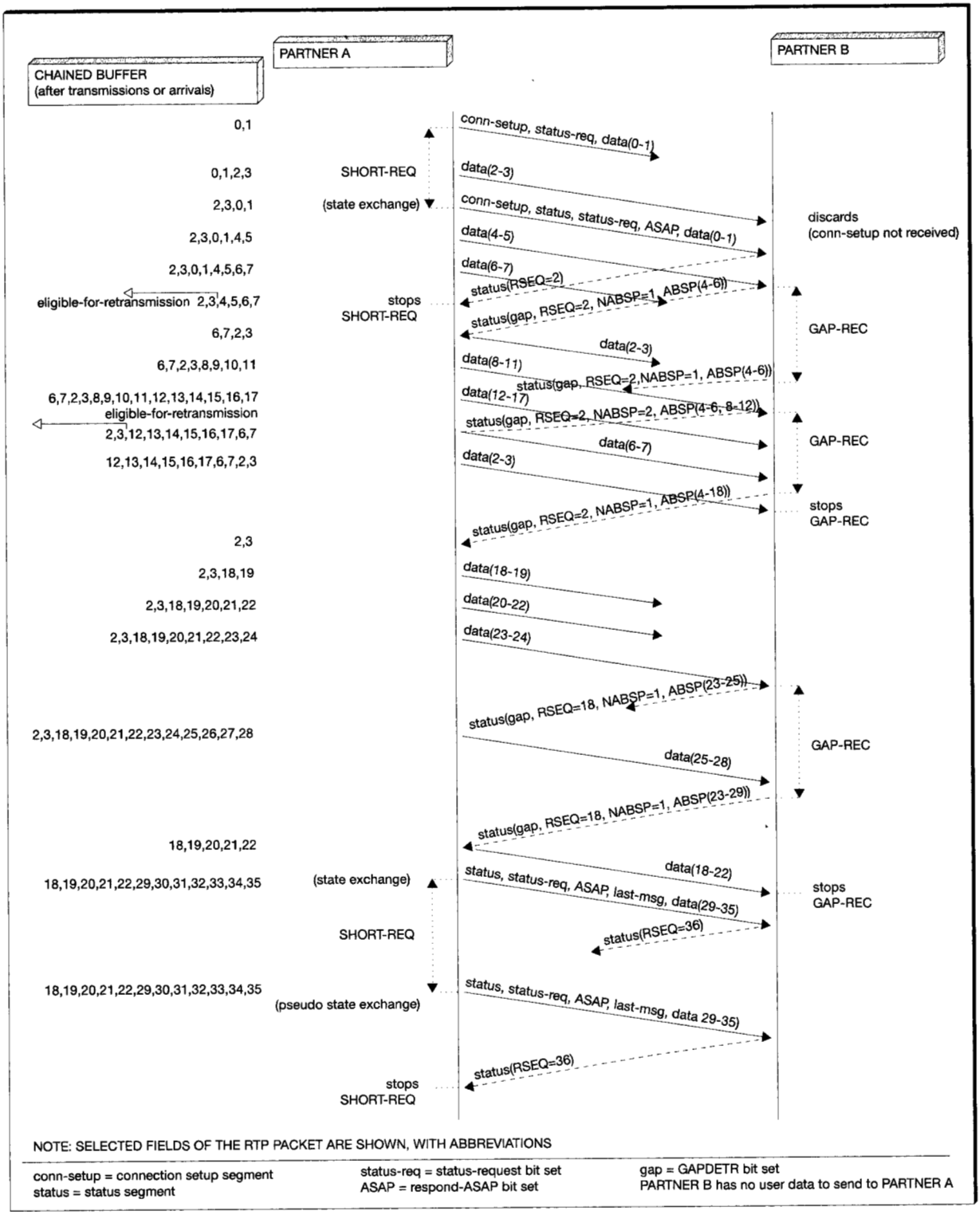
Whenever partner A receives a status segment with the GAPDETR bit set, it does the following: If the number-of-acknowledged-byte-span-pairs (NABSP) field is nonzero (regardless of whether the status segment is current or old), partner A retransmits all the eligible-for-retransmission message bytes between (and including) the byte sequence numbers RSEQ and acknowledged byte span pair begins (ABSPBEG) of the last ABSP. If the NABSP field is zero and the status segment is current, partner A retransmits all the message bytes from the byte sequence number RSEQ to the byte sequence number of the most recently transmitted message byte. When partner A retransmits message bytes, it initiates a state exchange with the last packet containing a retransmitted message byte if the NABSP field in the received status segment is zero. If the GAPDETR bit is set and the NABSP field is set to zero, this indicates that partner B is implementing the go-back-n scheme. The use of the chained buffer as described ensures that if both the partners implement the selective repeat scheme, no message byte will be retransmitted unless it is actually lost.

RTP uses the gap detection scheme to deduce that message bytes have been lost; however, this scheme does not work if a packet containing the last message (or the last segment of the last message) of a sequence of messages is lost. We refer to the message bytes contained in this packet as the "last message bytes." In order to ensure the correct delivery of the last message bytes, partner A initiates a state exchange with the transmission of the last message bytes. If the SHORT_REQ timer expires and a status segment acknowledging the receipt of the last message bytes has not been received, partner A retransmits the last message bytes and initiates a pseudo state exchange.[9]

Figure 4 shows an example of RTP error recovery when the selective repeat option is implemented.

Go-back-n. The scheme that will be described for the go-back-n option allows an implementation that supports selective repeat to communicate with an

Figure 4 Example of error recovery with selective repeat



implementation that supports go-back-n. The go-back-n scheme is the same as the selective repeat except as described here.

When partner B detects a gap (it receives a packet with a byte sequence number higher than expected), it does not buffer any message bytes that follow; it simply discards them. Then, partner B sends a packet containing a status segment with the GAPDETR bit set and the NABSP field not set to partner A. The status segment acknowledges the most recently received message bytes without an earlier gap. Unless in-sequence message bytes are received, any subsequent status segment sent by partner B has the GAPDETR bit set.

Unlike in the selective repeat scheme, partner A does not need to maintain a chained buffer in the go-back-n scheme.

Unlike in the selective repeat scheme, here partner A does not need to maintain a chained buffer. When partner A receives a current status segment with the GAPDETR bit set, it ignores the acknowledged byte span pairs (if any) and retransmits the message bytes from the byte sequence number RSEQ to the byte sequence number of the most recently transmitted message byte. Partner A initiates a state exchange with the last packet containing a retransmitted message byte.

Figure 5 shows an example of RTP error recovery when the go-back-n option is implemented.

Appendix B: Linear multicast mechanisms

This appendix describes the procedures and mechanisms that RTP uses to provide the three service options of RTP linear multicast (LM): unreliable multicast datagram (UMD), reliable multicast datagram (RMD), and reliable multicast transaction (RMT). The LM protocols are the same as the RTP point-to-point protocols except as specifically described here.

Unreliable multicast datagram. This service provides delivery of a single message from the client RTP to specified server RTPs with no reliability provision. When the client application has an unreliable message to multicast and requests the UMD service option, the client RTP constructs a packet with the no-retry bit set, a connection setup segment, and the message itself; then the client RTP multicasts the packet to the specified server RTPs and terminates the MPC. In this discussion, we assume that the packet size is large enough to hold a message and any required optional segments; otherwise, message segmentation is required as described previously.

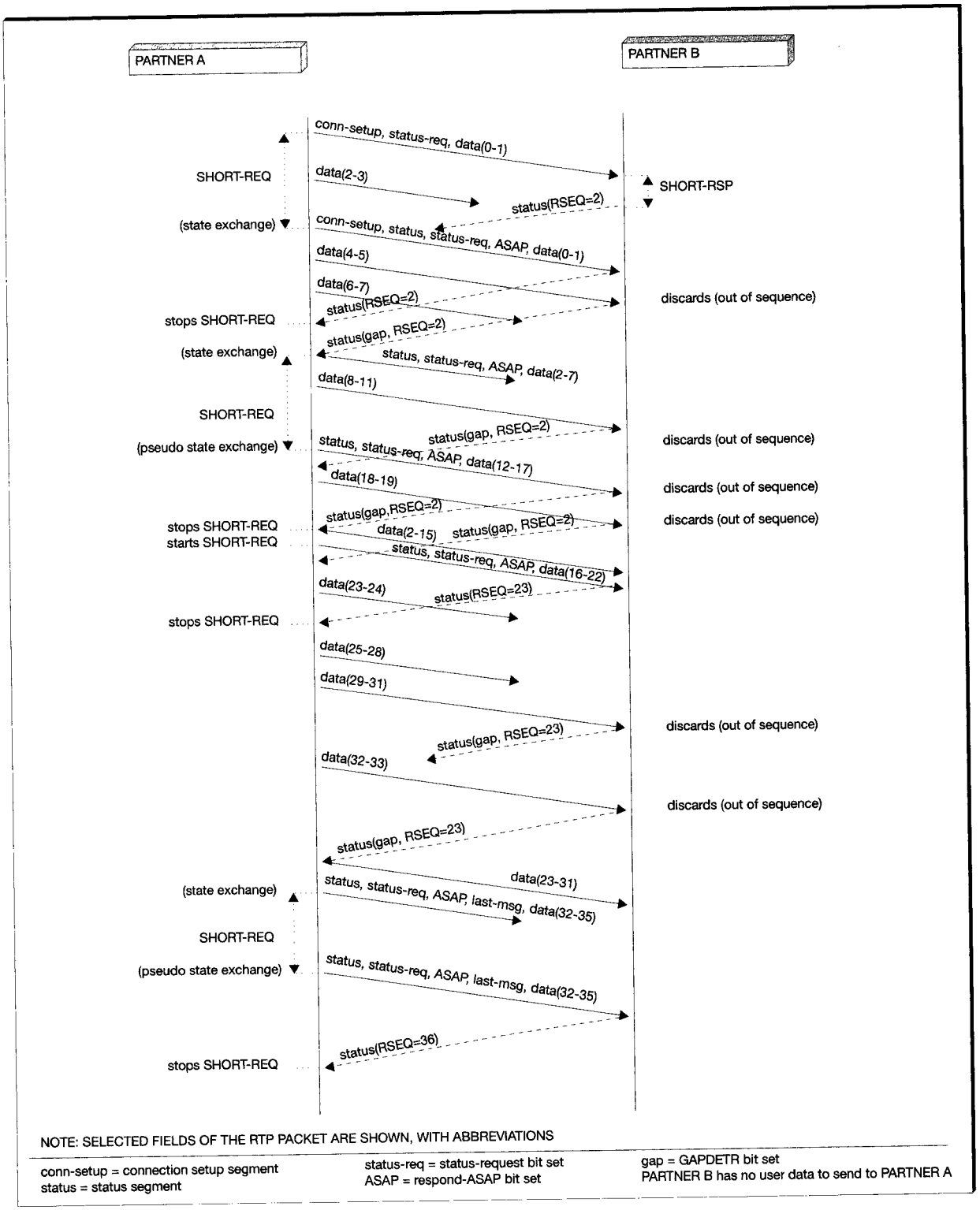
The server RTP waits for the arrival of a packet. When it receives a packet with the no-retry bit set and containing a user message, it delivers the application message to the server application and informs the server application that it received the message on an UMD MPC. Then the server RTP disconnects.

Reliable multicast datagram. This service provides reliable delivery of a single message from the client RTP to specified server RTPs. When the client application has a reliable message to multicast and requests the RMD service option, the client RTP constructs a packet containing the message with the status-request and respond-as-soon-as-possible (respond-ASAP) bits set, a connection setup segment, and a linear multicast segment with the linear-multicast-service-option (LMSO) bits set to RMD. It then multicasts the packet to the specified server RTPs and starts the SHORT_REQ timer.

If the SHORT_REQ timer expires, the packet is multicast to the server RTPs which have not yet acknowledged the receipt of the packet and the SHORT_REQ timer is restarted. The client RTP increases its retry count by one each time it transmits the packet. The send fails if the SHORT_REQ timer expires when the retry count is greater than the client MAX_RETRY. If the send fails, the client RTP reports a failure to the client application and disconnects.

If all the specified server RTPs have acknowledged the receipt of the packet, the client RTP stops the SHORT_REQ timer and informs the client application that the send completed.

Figure 5 Example of error recovery with go-back-n



The server RTP waits for the arrival of a packet. When it receives a packet with the LMSO bits set to RMD, the status-request and respond-ASAP bits set, and containing a message, it delivers the application message to the server application and informs the server application that it received the message on an RMD MPC. Because the status-request and respond-ASAP bits are set in the received packet, the server RTP immediately sends an acknowledgment (a packet containing a status segment with RSEQ set to one plus the sequence number of the end-of-message character of the received reliable message) to the client RTP.

When the server application requests RTP to disconnect, the server RTP has to dally (as explained in point-to-point connections) before it disconnects. This is because the client RTP may not have received the acknowledgment for its reliable message; in that case, the client RTP retransmits its reliable message.

Reliable multicast transaction. This service provides reliable delivery of a single message from the client RTP to specified server RTPs and delivery of one or more messages (only the last of which is reliable) from each server RTP to the client RTP. When the client application has a reliable message to multicast and requests the RMT service option, the client RTP constructs a packet containing the message with the status-request bit set and the respond-ASAP bit not set. The packet also includes a connection setup segment, a status segment (with ASEQ indicating the server RTP's send window allocated by the client RTP), and a linear multicast segment with the LMSO bits set to RMT. The client RTP then multicasts the packet to the specified server RTPs and starts the SHORT_REQ timer. The timer is stopped if the client RTP receives an acknowledgment from each server RTP.

If the SHORT_REQ timer expires, the packet is multicast again to the server RTPs that have not yet acknowledged the receipt of the packet. When the SHORT_REQ timer expires, the client RTP also sends a multicast acknowledgment (a packet containing a status segment with RSEQ set to the largest sequence number of the end-of-message characters of the received reliable messages, plus one) to the server RTPs that have sent their reliable messages and have acknowledged the receipt of the client RTP's message.

The client RTP increases its retry count by one each time it transmits (or retransmits) the packet. The send fails if the SHORT_REQ timer expires when the retry count is greater than the client MAX_RETRY. If the send fails, the client RTP sends a multicast acknowledgment to the server RTPs that have sent their reliable messages and have acknowledged the receipt of their client RTP's message. It then reports a failure to the client application and disconnects.

After the SHORT_REQ timer is turned off (after an acknowledgment is received from each server RTP) and before requesting disconnect, if the client RTP receives a reliable message from a server RTP, it will send a multicast acknowledgment to the server RTPs that have sent their reliable messages.

The transaction at the client RTP is considered completed if the client RTP has received an acknowledgment (for its reliable message) and it has received a reliable message (in a packet with the no-retry bit not set and the last-message bit set) from each server RTP. At this time, the client RTP sends a multicast acknowledgment to the server RTPs and informs the client application of the completion of the transaction.

When the client application requests to disconnect, the client RTP has to dally before it can terminate the MPC. This is because a server RTP may not have received the acknowledgment for its reliable message; in that case, the server RTP retransmits its reliable message.

The server RTP waits for the arrival of a packet. When it receives a packet with the LMSO bits set to RMT, with the status-request bit set and the respond-ASAP bit not set, and containing a message, it delivers the application message to the server application, informs the server application that it received the message on an RMT MPC, and starts its SHORT_RSP timer. (While the SHORT_RSP timer is running, it is not restarted if the server RTP receives another packet with the status-request bit set and the respond-ASAP bit not set.) If the SHORT_RSP timer expires, the server RTP sends an acknowledgment (a packet containing a status segment with RSEQ set to the sequence number of the end-of-message character of the received reliable message, plus one) to the client RTP.

After receiving the application message from the client application, the server application may send any number of unreliable messages. The server ap-

plication must then send a single reliable message. When the server application has a message to send, the server RTP constructs a packet containing the message. If the SHORT_RSP timer is running, the server RTP includes a status segment in the packet and stops the timer. If the message is reliable, the server RTP sets the status-request bit in the packet and starts the SHORT_REQ timer. The packet is then sent to the client RTP.

After sending an acknowledgment to the client RTP, if the server RTP receives a retransmission of the client RTP's reliable message, it sends another acknowledgment to the client RTP immediately.

If the SHORT_REQ timer expires, the server RTP retransmits the packet and restarts the timer. The server RTP increases its retry count by one each time it transmits the packet and starts (or restarts) its SHORT_REQ timer. The send (associated with the reliable message) fails if the SHORT_REQ timer expires when the retry count is greater than the server MAX_RETRY. If the send fails, the server RTP reports a failure to the server application and disconnects. If the server RTP receives an acknowledgment for its reliable message, it stops the SHORT_REQ timer and informs the server application that the send has completed.

Connection maintenance. Each server RTP has a connection inactivity timer that is used to detect loss of communications with the client RTP. For example, a transmission link could fail after a server RTP received the first packet of a segmented message from the client RTP. The timer is started (or restarted if running) each time the server RTP receives a packet from the client RTP. If it expires, the server RTP notifies the server application, and the server RTP disconnects. The timer is stopped when the server RTP receives a packet with the last-message bit set.

Appendix C: Unreliable multicast over trees mechanisms

This appendix describes the use of RTP multiparty connections (MPCs) to provide unreliable multicast over trees (UMOT) services. The UMOT protocols are the same as the RTP point-to-point protocols except as specifically described here.

RTP uses only the connection inactivity timer for the unreliable multicast over trees service (the other RTP timers are not used for this service). The connection inactivity timer is not used in the same way as in point-to-point RTP. The connection inactivity timer is required for the operation of each server RTP and is used to detect loss of communications with the client RTP. For example, a transmission link could fail after a server RTP received the first packet of a segmented message from the client RTP. The connection inactivity timer is started (or restarted if running) each time a packet is received from the client RTP. If it expires, the server application is notified and the server RTP disconnects.

The client RTP maintains a receive window allocation that is communicated via the ASEQ field in the status segment to server RTPs. The client RTP discards incoming application messages from server RTPs that do not fall within its receive window. A server RTP can only send application messages within the send window allocated by the client RTP. This provides a mechanism for the client application to perform window flow control on its side. If the client application does not want to receive application messages from server applications, it asks the client RTP to close its receive window. The client RTP closes its receive window by setting its RSEQ equal to ASEQ and multicasts a status segment in a packet to server RTPs. The client application can also ask the client RTP to allocate a new receive window. The client RTP allocates a new receive window by increasing its ASEQ (by the amount requested by the client application) and multicasts a status segment in a packet to server RTPs.

Unreliable multicast over trees MPCs have no separate MPC setup or dissolution. The client RTP sends the connection setup information with each message to allow server RTPs to join or leave the MPC at any time. Upon receipt of a message from the application, the client RTP constructs a packet containing an application message with the setup-packet and no-retry bits set and a connection setup segment. A status segment is also included in the packet if a receive window is allocated (application messages are expected). RSEQ and ASEQ in the status segment indicate the receive window allocation of the client RTP. Then the client RTP multicasts the packet to server RTPs.

When a server RTP receives a packet, with the setup-packet bit set, containing a connection setup segment, the server RTP simply ignores the setup information (the connection setup segment) if it has

already been received. When the server application has a message to send, the server RTP constructs a packet containing a message with the no-retry bit set and sends it to the client RTP. The server RTP can send a message only if its send window is not filled and the message size does not exceed the unfilled part of the send window. If a message cannot be sent, the server RTP informs the server application.

Cited references and notes

1. R. W. Watson, "The Delta-t Transport Protocol: Features and Experience," *Proceedings: 14th Conference on Local Computer Networks*, Minneapolis, MN (October 1989), pp. 399–407.
2. D. E. Comer, *Internetworking with TCP/IP*, Prentice-Hall, Inc., Englewood Cliffs, NJ (1991).
3. C. Fan, T. Luckenbach, and X. Xu, "Performance Comparison and Analysis of XTP and TCP/IP over the BERKOM Broadband ISDN Network," *Proceedings: IEEE INFOCOM '93*, March 30–April 1, 1993, San Francisco, CA (1993), pp. 1154–1161.
4. R. M. Sanders, "The Xpress Transfer Protocol (XTP)—A Tutorial," *Computer Communications Review* **20**, No. 5, 67–80 (October 1990).
5. M. H. Mguyen and M. Schwartz, "Reducing the Complexities of TCP for a High Speed Networking Environment," *Proceedings: IEEE INFOCOM '93*, March 30–April 1, 1993, San Francisco, CA (1993), pp. 1162–1169.
6. Networking Broadband Services (NBBS) Architecture Tutorial, GG24-4486-00, IBM Corporation (1995); available through IBM branch offices.
7. In packet switching, a datagram is a self-contained data packet sent independently through a network to its destination. It carries all of the information needed for routing from its origin to its destination.
8. I. Cidon, I. S. Gopol, S. Kutten, and M. L. Peters, "Distributed Tree Maintenance," *IBM Technical Disclosure Bulletin* **35**, 1A (June 1992).
9. A pseudo state exchange is the same as a state exchange except that an RTP machine does not examine the received state information from its partner for "freshness."

General reference

D. Bertsekas and R. Gallager, *Data Networks*, Prentice-Hall, Inc., Englewood Cliffs, NJ (1992).

Accepted for publication July 20, 1995.

Mohammad Peyravian *IBM Networking Hardware Division, P.O. Box 12195, Research Triangle Park, North Carolina 22709 (electronic mail: peyravn@ralvm6.vnet.ibm.com).* Dr. Peyravian is an advisory engineer in the Networking Architecture organization, where he is involved with protocol and algorithm design for high-speed networks. He received the B.S. and M.S. degrees from the University of Miami and the Ph.D. degree from the Georgia Institute of Technology in electrical engineering in 1987, 1989, and 1992, respectively. He does research in the areas of networking and telecommunications. He is the author of several papers and serves on the editorial advisory board of the *Computer Communications* journal. He has received several IBM invention and technical awards. He is a member of the Institute of Electrical and Electronics (IEEE) computer and communications societies, Tau Beta Pi, and Eta Kappa Nu.

Rachel A. Bodner *IBM AS/400 Division, 3605 Highway 52 North, Rochester, Minnesota 55901 (electronic mail: bodner@vnet. ibm.com).* Ms. Bodner is a staff programmer in the Networking Architecture organization, where she is involved with protocol and algorithm design for high-speed voice-, video-, and data-integrated networks. She received her B.S. in mathematics with a minor in computer science from North Carolina State University in 1989. She has coauthored several *IBM Technical Disclosure Bulletin* articles. She received an Outstanding Technical Achievement Award for her contributions to the Networking BroadBand Services architecture.

Chee-Seng Chow *IBM Research Division, Thomas J. Watson Research Center, 30 Saw Mill River Road, Hawthorne, New York 10532 (electronic mail: cschow@watson.ibm.com).* Dr. Chow received his bachelor's, master's, and doctoral degrees from the Massachusetts Institute of Technology. Since 1989 he has been a research staff member at the Thomas J. Watson Research Center working on high-speed communications. He is currently in the Network Security group working on proactive security and intellectual property rights management.

Marc A. Kaplan *IBM Research Division, Thomas J. Watson Research Center, P.O. Box 704, Yorktown Heights, New York 10598 (electronic mail: kaplan@watson.ibm.com).* Dr. Kaplan received a Ph.D. degree from Princeton University in electrical engineering and computer science in 1978. He received his B.A. degree from the College of Arts and Sciences at Cornell University in 1974. Dr. Kaplan has held various staff and management positions at IBM since the summer of 1978. He has designed and programmed file and security subsystems, operating system kernels, communications systems, and security systems. He architected, coded, and managed the software for the Paris/2-Planet-Orbit project. Dr. Kaplan is currently a research staff member at the Thomas J. Watson Research Center, working on electronic intellectual property rights management and protection.

Reprint Order No. G321-5586.