TN3270E Working Group
Internet Draft
Expiration Date: June, 1998

James Carmichael
Soumitra (Ronnie) Sarkar
IBM Corporation

December 1997

LU6.2 over TCP/IP
<draft-ietf-tn3270e-tcp62-00.txt>

Status of this Memo

Abstract

This draft describes IBM's support for running LU6.2 applications over a TCP/IP network. This support is provided using a subset of the X/Open architecture for Multiprotocol Transport Networking, which is a general protocol conversion architecture for running any application over any transport protocol. This draft is being distributed to members of the Internet community in order to solicit their reactions to the proposals contained in it. While the issues discussed may not be directly relevant to the research problems of the Internet, they may be interesting to a number of researchers and implementers.

Any questions or comments relative to the contents of this draft should be sent to the following Internet address: carmich@us.ibm.com.

Contents

1. INTRODUCTION

Applications that involve communications over a network are typically written using a given application programming interface (API) that ties it to a specific communications protocol, referred to as its "native" protocol. For example, Telnet uses the sockets API which ties it to TCP/IP, a version of DB2 uses the Advanced Peer to Peer Communications (APPC) API, the native protocol for which is LU6.2, Lan Requester uses the NetBEUI API and thus it's native protocol is NetBIOS, and so on.

Multiprotocol Transport Networking (MPTN) is an architecture for building a class of middleware systems that allow applications, originally designed to run on a given communications protocol, to run on any other protocol (i.e., run over a "nonnative" protocol) without change. Breaking the binding that typically exists between an application and a communications protocol enables users to reduce the number of protocols installed in their physical networks without giving up their favorite applications.

In contrast to encapsulation techniques such as Data Link Switching, where entire frames emitted by OSI layer 1 of a communications protocol are encapsulated as data over another communications protocol, MPTN uses a "protocol conversion" technique. TCP62 is the subset of MPTN architecture which supports programs written to APIs that depend on the LU6.2 protocol (e.g., APPC, CPIC) to run on a TCP/IP network.

[1]MPTN architecture addresses two separate problems - providing application independence from communications protocols, and the problem of network heterogeneity.

A node that performs protocol conversions at an end point, allowing an

application to run over a nonnative protocol without change, is called an "MPTN access node". A TCP62 node is an access node that converts LU6.2 to TCP/IP.

The network heterogeneity problem is addressed by an "MPTN gateway", which performs transport layer concatenation of heterogeneous networks. For example, a TCP62 gateway would allow an APPC application running on a TCP/IP network to communicate with an APPC application on a native SNA network. An MPTN gateway performs all the protocol conversions that an access node performs, and additional protocol conversions to hide the nonnative network (in the TCP62 case, TCP/IP) from the native network's control flows (e.g., APPN control point protocols). MPTN gateway technology is not an X/Open standard. This draft only covers the access node function for LU6.2 over TCP/IP.

2. A HIGH LEVEL OVERVIEW OF TCP62 FUNCTION

To implement TCP62 function on a given platform, the following issues need to be considered.

1. The LU6.2 stack on that platform needs to be modified so that transport layer functions of an LU6.2 session are provided by a TCP connection, after the session traffic has been processed by protocol conversion software.

   Since transport layer functions required by the LU6.2 stack do not exactly match the characteristics of TCP connections, "protocol compensations" have to be performed by the conversion code. Definition of the formats and protocols for performing such compensations is the primary focus of this draft.

   An important issue that needs to be addressed is - what constitutes LU6.2 transport layer function. A secondary, but related issue is - which LU6.2 headers need to flow across a TCP/IP network for higher layer (OSI layer 5 and above) functions.
2. Protocols associated with LU6.2, such as Advance Peer to Peer Networking (APPN) control point protocols, as well as LU6.2 session level pacing protocols, must be prevented from flowing on the TCP/IP network.
3. Since TCP62 allows LU6.2 applications to run unchanged over a TCP/IP network, LU6.2 network resource names used by the latter applications have to be mapped to the IP addresses of the machines where those resources reside. This is referred to as the "address mapping" problem.

The figure below illustrates logical components in a node running TCP62.

```
                       ---------------
                       |  APPC/CPIC  |
                       | application |
                       ---------------
                              ^
                              |
                              v
                       ---------------
                       |   LU6.2     |
                       |  protocol   |<---
                       |    stack    |   | Transport layer
                       ---------------   | requests
                                         v
                            -----------------
                            |    Protocol   |
                            | compensations |
                            |       +       |
                            |    Address    |
                            |    mapping    |
                            -----------------
                                    ^
                                    |
                                    v
                            =================
                            == sockets API ==
                            =================
                            -----------------
                            |    TCP/IP     |
                            |    protocol   |
                            |    stack      |
                            -----------------
```

Figure 1. High Level Overview of the Components of a TCP62 Node

2.1. **Protocol Compensations Required for TCP62**

TCP62 maps LU6.2 sessions to TCP connections. It is a one-to-one mapping. LU6.2 sessions provide connection-oriented semantics, ensuring in-order, reliable delivery of data. OSI layer 5 and above functionality is provided with half-duplex session management semantics and transaction support (sync point). Since LU6.2 session semantics do not exactly match the functions provided by a TCP connection, TCP62 code must compensate for the differences. The functional differences between the session semantics of each protocol is outlined below.

   1. Connection setup with connection data:

When an APPC application wants to communicate with a partner application, it requests that an APPC "conversation" for a given class of service (COS) be set up to a partner LU, where the partner application is identified by a "transaction program (TP) name". The APPC stack in turn checks if an LU6.2 "session" for that COS already exists to the partner. If so, then the conversation setup involves flowing the appropriate LU6.2 format (FMH5) on that session as control data. Otherwise, the session must be set up over an MPTN connection using MPTN protocols.

LU6.2 session setup involves flowing a BIND (a set of bits that define session parameters) to the session partner, which in OSI terms constitutes "connection data".
The session partner returns a BIND response (connection response data), and if it is acceptable to the node that initiated the connection request, session setup is complete. Since the TCP connection setup protocol does not require connection data, this is an area that will require protocol compensation.

2. Record-oriented traffic

APPC conversations guarantee that units of data sent by APPC applications will be delivered to the session partner with record boundaries kept intact. This should be contrasted with TCP, which is a stream-oriented transport layer protocol.

3. Expedited Data:

In MPTN architecture, expedited data is defined as "out-of-band" data on a connection that can bypass the normal queueing mechanisms that "normal" data on that connection is subjected to. Expedited data on a connection can therefore bypass (i.e., arrive at the destination before) normal data that was sent before it, but must not arrive at the destination any later than normal data that was sent after it.

The LU6.2 protocol stack requires support for expedited data for the following two reasons:

- In older versions of the APPC API which supports half-duplex conversations only, applications cannot send expedited data, but the protocol stack itself sends expedited data that needs to bypass normal data which can be held up due to queueing LU6.2 congestion control.

- In full duplex APPC APIs (including CPIC), applications can send expedited data with the same properties that the APPC protocol depends on.

Expedited data in LU6.2 can be more than one byte long, and has more general properties than those provided by the TCP (1-byte) urgent data mechanism, making this another area where protocol compensations will be required of a TCP transport provider,

4. Session Outage Notification:

APPC sync point functions (e.g., which provide 2-phase commit like capability) depend on the APPC API to provide them with immediate notification when a session is disrupted. In LU6.2, such notification is provided by the Data Link Control (DLC) layer of the stack, which performs LLC-2 level "keepalive" protocols with the adjacent link station that constitutes the next hop in an end-to-end LU6.2 session.

TCP/IP implements a keepalive protocol on a per connection basis, but on most implementations (at the time when TCP62 was designed), the keepalive timers were quite high (e.g., 2 hours) and were not configurable. This led to the definition of a keepalive compensation for this function, which needs to be implemented by TCP62.

5. Duplex-Abortive Session Termination with Termination Data

In MPTN architecture, a connection termination request is classified as duplex if transport user on one end of the connection can terminate both ends of the full-duplex connection. The connection termination is classified as abortive if no guarantees are required to ensure that data in flight needs to reach the other end of the connection, when the termination request is received by the protocol stack.

APPC applications view connections as conversations, whereas, the LU6.2 stack's view of a connection is a session. Since LU6.2 sessions are normally terminated only when no conversations are active, the requirement for abortive close seems quite natural. Session termination with a conversation active is a severe enough condition where any attempts to preserve the integrity of data in flight would be meaningless.

An LU6.2 session is terminated by exchanging UNBIND and UNBIND response formats on the session. In MPTN (as well as OSI) parlance, this is known as termination data. Since TCP connections do not provide the exchange of termination data during connection shutdown, this is yet another area where a compensation is required.

2.1.1. LU6.2 Headers that Flow Over TCP/IP

Since LU6.2 transport function in TCP62 is performed by TCP/IP, only those LU6.2 headers that are required by higher layers of the LU6.2 stack need to flow over a TCP connection. For all LU6.2 formats that flow over TCP connections, only the following headers are included:

- only the sequence number field of the LU6.2 transmission header (TH)
- the entire request/response header (RH), except when reassembly of LU6.2 request/response units (RUs) is being performed by TCP62.

While it is hard to exactly map SNA protocol stack functions to OSI transport and session layer functions, it is a reasonable approximation that the TH represents transport layer information while the RH contains session layer information needed to implement functions such as half-duplex conversations. The sequence number field in the TH is used to implement acknowledged data transmissions in APPC and CPIC API, and since that is not a typical transport layer function in protocols other than SNA, a compensation was not provided for it, but instead, the LU6.2 implementation of that function is preserved - thus preserving a portion of the transport header.

2.1.2. A Sample Flow

The example below illustrates the flows that take place when an APPC/CPIC application on Node A requests a conversation with an application on Node B. The LU6.2 stack finds no existing session to reuse, and therefore creates one. The request goes through the TCP62 code, and the LU6.2 session is mapped to a TCP connection. Application data flows on the LU6.2 session, and as payload on the TCP connection (and sometimes, as payloads of UDP datagrams) after MPTN headers have been added to indicate the protocol compensations being performed. Finally, the conversation ends, and the session is terminated (typically, LU6.2 sessions stay up and get reused for new conversations; the session termination case is used to demonstrate the entire life cycle of the connection).

```
       Node A                                          Node B
   ---------------                                 ----------------
   |  APPC/CPIC  |                                 |   APPC/CPIC  |
   | application |                                 |  application |
   |=============|                                 |==============|
   |   LU6.2     |                                 |    LU6.2     |
   |   stack     |                                 |    stack     |
   |=============|                                 |==============|
   |    TCP62    |                                 |     TCP62    |
   |=============|                                 |==============|
   |   TCP /IP   |                                 |    TCP/IP    |
   ---------------                                 ----------------

   Step 0: Perform the address mapping to discover the partner
           node's IP address, given the LU name that the application
           uses. The details of that step are omitted.

   Step 1: Setup native TCP connection to partner node
               <================================>

   Step 2: Exchange MPTN connection setup packets as data on TCP
           connection, thus mapping an APPC session to a TCP connection.

               IP header, TCP header, MPTN_Connect (incl. BIND image)
               -------------------------------->
               IP header, TCP header, MPTN_Connect RSP (incl. BIND RSP)
               <--------------------------------

   Step 3: Exchange LU6.2 (connection-oriented) application data over
           TCP connection, using TCP62 protocol compensation headers.
           Occasionally, UDP datagrams need to be used for certain
           compensations, as described in a later section.

               IP header, TCP/UDP header, compensation header,
               SNA TH subset + RH, application/protocol data
               <================================>

   Step 4: Terminate SNA session using TCP62 protocols.
           The termination protocols use UDP only.

               IP header, UDP header, protocol compensation header,
               SNA TH subset + RH, application/protocol data
               <================================>
```

Figure 2. High Level Overview of TCP62 Protocols

2.2. LU6.2 and Related Protocols to Disable When Running TCP62

Since TCP62 was designed to be a protocol conversion approach instead of a protocol encapsulation approach, every effort was made to filter out aspects of LU6.2 and related protocols for which analogous functions exist on a TCP/IP network. The two categories of protocols to be disabled in a node running TCP62 are

1. APPN control point protocols.
2. LU6.2 pacing protocols.

APPN control point protocols serve the same function as routing protocols in a TCP/IP network. These protocols implement a link state routing algorithm (similar to OSPF), where special nodes called network nodes (which form the APPN network backbone) continuously exchange topology information, and use that information to compute an optimal route between two logical units (LUs).

LU6.2 uses a window based flow control mechanism, which is conceptually similar to TCP/IP's sliding window protocol. LU6.2 protocols use an "adaptive pacing" mechanism, where the stack at the sending end of a connection requests additional windows (one for each record that it will send) from the receiver by setting a "pacing request" bit in the SNA Record Header (RH). The stack at the receiving end of the session sends an Isolated Pacing Message (IPM), based on the buffer capacity in the node, to indicate how many more records the transmitting side can send it.

Since TCP/IP already uses an various IP routing protocols that allows any node on an IP network to reach any other node with an IP address on that network, TCP62 prohibits APPN protocols (the routing protocol associated with APPC) from running over TCP/IP, since that would amount to running two separate routing protocols on the same network. The address mapping compensation is used to map partner LU names to partner IP addresses, and existing IP routing mechanisms (and ARP) are sufficient to complete the rest of the steps leading to native SNA connection setup.

LU6.2 flow control protocols are disallowed when running TCP62. In particular, the pacing request bit of the LU6.2 RH must never be set when sent over a TCP connection. In addition, IPMs must also be prevented from flowing over TCP/IP. The rationale for this decision is to prevent one set of pacing protocols from being run of top of another. A TCP62 implementation must map the APPC adaptive pacing protocol to TCP/IP's flow control protocol. This is described in greater detail in Section 4.2, "Pacing".

2.3. TCP62 Address Mapping

Distributed applications use names to identify communication partners. Different communications protocols use diverse naming conventions to identify partner nodes, and applications within those nodes. APPC applications use LU6.2 names to identify a logical units in nodes. LU names are of the form

```
    NETWORK-ID.LOGICAL-UNIT-NAME
```

where both the network-id and logical-unit-name parts can be up to 8 bytes long and use a subset of the EBCDIC character set. A transaction program (TP) name, which can be up to 64 bytes long, is used to identify the application in the partner node. A socket application uses a 4-byte IP addresses to identify partner node, and a port number to identify the partner application.

Address mapping can be thought of as a type of compensation to be provided, not due to a mismatch in transport protocol function, but due to a mismatch in naming conventions used in different protocols to identify communications partners. Since an APPC/CPIC application uses an LU name to identify its partner, this has to be translated to an IP address by TCP62 so that communication with the node where the partner LU resides can take place over TCP/IP.

In LU6.2, only the partner LU name is used when a session is set up. The TP name is exchanged in a later flow. Therefore, a solution to the address mapping problem in TCP62 requires a mechanism for mapping a partner LU name to the partner IP address. Address mapping is performed using the TCP/IP Domain Name System (DNS), which consists of a resolver (client), which queries a domain name server or a local HOSTS file to map a domain name to an IP address. Address mapping is TCP62 is implemented as follows:

1. For each LU6.2 name that is a potential partner of an APPC application running over TCP/IP, the name is converted to a domain name using the following algorithm:

   - the EBCDIC characters are converted to ASCII characters
   - the ASCII character name, which is of the form

         NETWORK-ID.LU-NAME

     is converted to the form

         LU-NAME.NETWORK-ID

     by swapping the network and LU-name portions on either side of the period.

   - an SNASUFFIX (a configured value) that is common to all TCP62 nodes in the user's TCP/IP network is added to the above

name.

The SNASUFFIX is of the form A.B.C such that it adheres to the rules of a valid TCP/IP domain name suffix.

For example, if the LU name (in ASCII) is NET1.LU1, and the SNASUFFIX value for the customer network is dept1.company2.com, then the domain name derived from the LU name would be:

    LU1.NET1.dept1.company2.com

2. The domain name is mapped to the IP address of the node where that LU resides (in a TCP62) node, by either entering the mapping in the database of a domain name server that serves the domain dept2.company1.com, or in a HOSTS file that would need to be distributed to all TCP62 nodes in the network.

3. When an LU6.2 stack in a TCP62 node requests an LU6.2 session to a partner LU, the TCP62 address mapping code must do the following:

   - apply the algorithm described in step 1 above to map the partner LU name to a domain name
   - query the DNS (using a standard sockets API call such as gethostbyname) which will return the IP address mapped to that domain name
   - use the IP address that was returned to set up a TCP connection to that node, in preparation for performing the connection setup protocols to that node.

3. Protocol Compensations

The previous section introduced the need for protocol compensations. This section describes how each compensation is performed. The formats of MPTN flows that are used to perform these compensations is described in Section 5, "TCP62 Formats".

In TCP62, one TCP connection is used to map an LU6.2 session. Compensations are implemented by flowing MPTN headers, usually followed by payloads that contain SNA packets. The MPTN header conveys to the receiver the nature of the compensation, allowing it to take appropriate action. Some MPTN headers flow on the TCP connection representing the session, whereas others flow as payloads of UDP datagrams, either to bypass TCP flow control (e.g., for expedited data), or because the compensation is node-level rather than session level (e.g., for implementing a keepalive protocol).

The life cycle of an SNA session can be summarized as follows:

- A session is set up by first creating a TCP connection to the node hosting the partner LU, whose IP address is determined using address mapping. The first packet on the connection is an MPTN_Connect, a portion of which contains the LU6.2 BIND image. The partner TCP62 node sends an MPTN_Connect response packet back, which includes the BIND response, and the connection set up is complete.
- Regular data transfers now begin. Data that flows on the connection always consists of a 1-byte header that describes the compensation. This is typically followed by SNA headers and data.

  Expedited data flows using a more complex compensation mechanism. The data is first sent on the connection, but if an expedited data acknowledgement packet is not received in a fixed time (e.g., 3 seconds in IBM implementations), the expedited data is sent to the partner in an "out-of-band" (OOB) datagram. OOB datagrams use a more complex format and flow as UDP datagrams.

  All SNA sessions require session outage notification compensation, which is performed using "keepalive" datagrams. Keepalive datagrams do not have any payload and also flow as UDP datagrams. The keepalive compensation is performed on a node-pair basis, as long as at least one LU6.2 session exists between those nodes.

- Finally, if the session is to be terminated, the UNBIND image is sent to the partner node using the expedited data mechanism.

3.1. Connection Establishment

When establishing a LU6.2 session over TCP/IP, the normal TCP connection establishment occurs first. The destination address is a combination of the IP address obtained using address mapping, and the well-known port utilized by MPTN. The values for the well-known ports are 397 for TCP and 397 for UDP. After the TCP/IP connection has been established, the two nodes exchange an MPTN_Connect command and response in order to set up an MPTN connection. The initiating node sends the MPTN_Connect command, including addressing information, connection characteristics, and any connection data required. For LU6.2, the connection data is the BIND RU. For detailed information about the MPTN_Connect and MPTN_Connect response as they apply to LU6.2 over TCP/IP, please refer to Section 5.3.1, "MPTN_Connect".

After processing the MPTN_Connect, the target node will send an MPTN_Connect response. This response may include some changes to the connection characteristics. The negotiable connection characteristics are: maximum record length, maximum expedited record length, and

maximum length of connection termination data. These connection characteristics may be negotiated down. The characteristics sent in the MPTN_Connect response are the ones used on the connection.

In the MPTN_Connect command and response, there is a correlator suffix field. This correlator suffix, when combined with the destination and source MPTN addresses, must be unique for all time.

3.2. Data Transfer

When LU6.2 session data is sent, the stack builds a TH and an RH which precedes the payload. The SNA headers are manipulated by the TCP62 code so that only the sequence number part of the TH, and the RH with the pacing request bit set to 0, flows on the TCP connection. On the receiving node, the TCP62 code must rebuild the part of the TH that was truncated by the sending node, and must also set the pacing bit on the RH based on the implementation of pacing as described in Section 4.2, "Pacing".

SNA session data is prefixed with a one-byte MPTN compensation header that indicates the type of compensation being performed. When the data flows over TCP, the compensation header is preceded by a four-byte length field that indicates the length of the SNA data (including headers), and also includes the sizes of the MPTN header and length fields (5 bytes).

3.2.1. Nonexpedited Data

In TCP62, nonexpedited (or normal) data needs record-over-stream compensation, since SNA honors record boundaries while TCP/IP is stream-oriented. Normal data always flows over TCP, subject to the flow control of the TCP connection. The one-byte compensation header indicates that it is nonexpedited data with record boundaries preserved, and the TCP62 code on the receiving node does not deliver it to the local stack until the entire record arrives.

3.2.2. Expedited Data

3.2.2.1. Sending Expedited Data

LU6.2 requires the ability to send expedited data that fulfills two requirements:

1) The expedited data must arrive before data sent later on the connection.

2) The expedited data must be able to bypass normal data queues, which may be backed up due to congestion.

Both the LU6.2 protocol stack, as well applications that use the full duplex APPC API have the ability to send expedited data even if the receiver is not receiving normal data.

In order to provide for these requirements, an MPTN compensation is provided that sends data on the normal connection (to meet requirement 1, listed above), and, if needed, as a datagram (to meet requirement 2). The receiver processes the version that arrives first, and discards the other.

TCP/IP supports urgent data, but it's features are not rich enough to meet the requirements of LU6.2.

- While TCP/IP support urgent data, there is no guarantee that the urgent data will be sent if the window size on the TCP connection is zero. Under zero-window conditions, newer releases of BSD code send a TCP packet without any payload, but with the urgent bit set, to inform the partner about the availability of urgent data.
  The partner can read all the normal data that precedes the urgent data (the process of receiving the normal data will increase the window size, enabling the sender to send data). However, when TCP62 was first designed, there was no indication that many TCP/IP implementations on Wintel systems had this fix.

- The above fix, even if present, is not sufficient for LU6.2 use. The TCP urgent data mechanism only handles one byte of urgent data, though additional urgent data can be handled if there is an a priori agreement between the sender and the receiver about the length. Since LU6.2 expedited data can have arbitrary lengths, any scheme based on predefined lengths is infeasible.

- Finally, the TCP urgent data mechanism only marks the most recently sent urgent data. If the receiver does not process urgent data #1 by the time urgent data #2 is sent, all indications that data #1 was urgent are lost.

Given these limitations of the TCP urgent data mechanism, the expedited data compensation designed for TCP62 is based on the assumption that TCP has no expedited support at all.

Expedited data received from the LU6.2 stack is sent on the TCP connection in the same way as normal data (with the LU6.2 headers doctored, and preceded by a length and compensation header field),

except that the one-byte header field indicates that the data is expedited. Expedited data requires an acknowledgement, which consists of a length field and a one-byte compensation header that indicates that it is an acknowledgement. There is no payload associated with this, and so the length field should have a value of 5. The receiving TCP62 node should send this acknowledgement as soon as it receives the expedited data. Note that since the expedited data is sent on the TCP connection, it is subject to TCP flow control, and may take a considerable amount of time to reach the partner. Since this might happen, the compensation mechanism uses a timer to wait for the acknowledgment, and if it does not receive the acknowledgement in a reasonable amount of time (the default is 3 seconds in IBM implementations), then the data is sent using MPTN OOB datagrams, whose format is described in Section 5.3.3, "MPTN_DG_OOB_Data".

When expedited data is sent as an OOB datagram, it appears as a payload of an UDP datagram, whose destination address is the IP address of the node hosting the partner LU, and the UDP port number is 397. The data portion of the OOB datagram only consists of the MPTN header and the SNA header subset and payload. Since UDP datagrams are not stream-oriented but preserve record boundaries, the 4-byte length field is not required.

When expedited data is sent on an OOB datagram, the partner is expected to respond with an OOB reply datagram which has no payload. If this reply is not received in a reasonable amount of time (e.g., 10 seconds is a default value on some IBM implementations, but the value can be customized), the OOB datagram is retransmitted. If five retransmissions yield no response, the connection is assumed to be dead and is torn down.

In addition to the data being sent, the MPTN_DG_OOB_Data format includes a connection correlator used to identify the associated connection, and a connection sequence number that uniquely identifies that piece of expedited data on the connection. These fields are described next.

3.2.2.1.1. Sequence Numbers in OOB Datagrams

Since expedited data that flows in an OOB datagram may reach the partner in two different ways, the sequence number field allows the receiving TCP62 node to discard whichever copy reaches it last. Note that when the expedited data is first sent on the TCP connection, no sequence number is included. Instead, the receiving TCP62 node is expected to keep a count - which is sufficient because TCP delivers data in order, and without loss. A sequence number field is required in the OOB datagram because UDP datagrams can be lost and be delivered out of order.

The connection sequence numbers start at zero and increment after each

expedited message is sent. The sequence number field wraps (reset to zero) at 65,535, which is sufficient to uniquely identify a given record sent in an expedited fashion, assuming a reasonable amount of loss and reordering of UDP datagrams. Retransmission of a previous expedited message is not counted as a separate event.

3.2.2.1.2. Correlating an OOB Datagram to a Connection

The connection correlator in an OOB datagram consists of two parts: the correlator address, which is a field that was assigned by the TCP62 node that initiated the connection (and flowed on the MPTN_Connect packet), and the correlator suffix, which is a value unique to SNA. The two parts are concatenated to form the connection correlator.

There are two ways to correlate the MPTN_DG_OOB_Data to the associated connection; the choice is up to the initiator of the format:

1) Using the connection correlator field only.

2) Using the connection correlator field and the optional receiver connection alias field.

The connection correlator is recognized by both sides without any extra protocol being involved. The receiver connection alias is a local identifier that can be used by the destination for a quicker identification of the connection. The receiver connection alias on an inbound OOB datagram can be used to identify the connection in a faster manner than performing a match on the connection correlator. The connection alias could be a connection control block pointer, a socket number, or any other field that is deemed suitable by an implementation.

In the first MPTN_DG_OOB_Data sent for the connection, the sender can provide its own local identifier (in the sender connection alias field) in addition to the connection correlator. The receiver locates the connection using the connection correlator and stores the sender connection alias so that it knows the value to be returned in the receiver connection alias field in the future.

Whenever a TCP62 node builds an MPTN_OOB_DG_Data format to send to its connection partner, it should include a receiver connection alias, if available. Inclusion of the receiver connection alias is recommended for performance reasons.

The sender of an OOB datagram should continue to include its sender connection alias until it receives an indication from the target that it has been received, either in a positive response to an MPTN_DG_OOB_Data sent, or in an MPTN_DG_OOB_Data command from the partner with the

receiver connection alias containing its local id.

3.2.2.2. Receiving Expedited Data

The target of the expedited data transfer must respond to each instance of the expedited data received with an appropriate acknowledgement, either with an MPTN Header marked as an expedited acknowledgement which is sent on the TCP connection, or an MPTN_DG_OOB_Data response packet, which is sent using UDP.

In the case of the MPTN_DG_OOB_Data response, the connection sequence number is included in the MPTN_DG_OOB_Data response, and informs the partner that the data has been received and any retransmissions should be stopped.

3.3. MPTN Keepalive Protocol

TCP62 nodes must implement a keepalive protocol because TCP connections do not provide session outage notification (SON) in a timely manner. Even though TCP provides a per connection KEEPALIVE facility, the typical values of TCP keepalive timers (the period of inactivity after which liveness checking is initiated) are much larger than SNA's SON requirements. Note that in many current TCP implementations, the KEEPALIVE timer is settable on a per connection basis, but since this was not true when TCP62 was first designed, this compensation was deemed necessary.

3.3.1. The Problem of Out-of-Sync Connections

When two access nodes have a number of MPTN connections between them (ignoring connections that are coming up or are being taken down), one of two things can happen to cause them to have different views of which connections they have with each other:

- One of the access nodes crashes. This does not allow any connection termination flows to reach the partner node to allow the connections to be taken down. None of the TCP connections are taken down either. If there is no activity on any of the sessions, the two sides will have an inconsistent view of which LU6.2 sessions they have with each other until the TCP KEEPALIVE mechanism kicks in.

- An intermediate router in the TCP connection between the two TCP62 nodes goes down for a while. During that period, each of

the two nodes can independently close one or more LU6.2 sessions that they have with the partner. However, because of the dead router, neither the TCP62 flows nor any TCP connection termination flows can reach the partner. This would result in the two nodes having an inconsistent view of the connections they have with each other.

When two active TCP62 nodes have inconsistent views of the number of LU6.2 sessions they have with each other, a new LU6.2 session set up request initiated by one node (which believes that the number of existing sessions is within limits) may be rejected by the partner node (which believes that session limits have already been reached).

The above situation is a direct result of the fact that while the router was down, the TCP keepalive mechanism did not get triggered. The assumption here is that the router went down and came back up before the TCP keepalive timer expired. Since TCP keepalive timers are often set to an hour or more by default, and are not changeable in many implementations, this is not an unrealistic scenario.

If the TCP keepalive timer was user settable, this problem could be avoided by setting the timer value that is smaller the time required for a router or an access node to be restarted after a crash (e.g., 2 minutes). However, most implementations do not allow this timer to be changed by the user. The alternate solution is to implement an MPTN keepalive protocol. The protocol is described in the next section.

3.3.2. A Keepalive Protocol Using Datagrams

The MPTN keepalive protocol is executed by two TCP62 nodes that have one or more LU6.2 sessions with each other. LU6.2 sessions require SON, and while SON is a session level characteristic, the keepalive protocol is optimized to operate at a node-pair level. SON is a transport user characteristic that flows on the MPTN_Connect packet (see Section 5.3.1.1.3, "User Characteristics") and both TCP62 nodes are thus aware of when the keepalive protocol should be performed. The keepalive protocol between a pair of nodes is stopped when the last LU6.2 session between them is terminated.

The keepalive protocol involves sending a keepalive datagram if an inactivity timer expires, and waiting for a response.

- When the first LU6.2 session between a pair of TCP62 nodes is set up, an inactivity timer is started for that partner node. The value of the timer should be user settable.

- When any data is received from the partner node - such as a

connection request, data on an existing connection, an OOB datagram, or a keepalive datagram - the timer is reset. However, if no data is received from that partner before the timer expires, the MPTN keepalive protocol is initiated with that partner by sending it a keepalive datagram. Like OOB datagrams, keepalive datagrams flow as payloads of UDP datagrams and are sent to port 397, the MPTN well-known datagram port number. Keepalive datagrams never include any LU6.2 payload.

- Once the keepalive protocol is initiated, a keepalive datagram is sent to the partner once every X seconds, where X is a value that can be configured by the user. When a TCP62 node receives a keepalive datagram, it must send a keepalive datagram RSP packet back to the sender. If a node does not receive a response to a keepalive datagram, or receive some other communication from the partner node after having sent the datagram 5 times, it concludes that the partner is either dead or is unreachable, and all LU6.2 sessions to that partner are terminated.

To cover the case where a node goes down and comes back up before the KEEPALIVE protocol is able to detect the outage, an optional node initialization ID field is included as part of the MPTN_Connect command. This field contains a value that uniquely identifies the instance of the sending node, e.g., a time stamp corresponding to the time the node was initialized. The receiving node keeps this value, and compares it with future incoming node initialization IDs from the same partner. If the values are different, the partner has gone down and recovered. Thus, all sessions with the partner that require SON should be cleaned up.

3.4. Connection Termination

When an LU6.2 session is terminated, an RU containing an UNBIND image is sent to the partner. A session termination request assumes that both ends of the session will be terminated, and any data in flight can be dropped. Since APPC applications use conversations on top of sessions as application level connections, and since a session is rarely brought down when a conversation is using it, these semantics make sense.

In LU6.2 protocols, an UNBIND image always flows as expedited data, since it should not be held up by session level pacing. Therefore, in TCP62, the transmission of the UNBIND request must use expedited data compensation. Unlike all other expedited data, the UNBIND is only sent as an OOB datagram, without first making an attempt to send it on the connection first. The regular expedited data compensation mechanism is designed to deal with the fact that the data can reach the partner node in two different ways. In this case, since the action the receiver will take is to terminate the connection, it was decided that a simpler

option would be to flow the termination request using one mechanism only. And since using the TCP connection is potentially delay-prone, the OOB option was chosen.

When an UNBIND image is sent to the partner as an OOB datagram, its payload consists of a 1-byte compensation header that indicates duplex-abortive compensation, and the payload in the UNBIND preceded by the TH subset and the RH. As with regular expedited data, the payload does not include the 4-byte length field. When the TCP62 code sends the termination request, it should discard any data that was queued for transmission on that session, and should disable further reception of inbound data on that connection.

The TCP62 node that receives the UNBIND request passes it up to the local SNA stack, and terminates the TCP connection after discarding any session data that was queued for transmission on that connection. The TCP connection termination uses the standard socket "close" call, and represents orderly shutdown of one end of the connection. When the local SNA stack sends an UNBIND response, the TCP62 code sends an OOB response datagram back to the sender to indicate that the session termination on that node is complete. The inclusion of the UNBIND response image in the OOB datagram is optional.

When the node that initiated the connection termination request receives the OOB datagram response, it should inform the SNA stack (manufacturing an UNBIND image if one was not included in the response datagram), and finally, it should terminate it's end of the TCP connection using the orderly termination semantics of the socket "close" call.

4. Other Protocol Issues Not Related to Compensations

4.1. Segmentation and Reassembly of APPC Request/Response Units

In the APPC protocol, a request/response unit (RU) represents a unit of data whose size (the RU size) is typically negotiated by all nodes in the session path based on the smallest frame that a DLC in the session path can transmit on a link. Irrespective of the size of the record that an application sends across the APPC API, the largest record that flows as the payload of an LU6.2 packet cannot be bigger than the RU size. This ensures that when a negotiated RU size is used, no segmentation or reassembly logic will be required in any hop of the session path.

When an APPN node in the session path does not implement RU size negotiation function, segmentation and reassembly becomes inevitable for

that session when run natively. When LU6.2 sessions are run over TCP/IP connections using MPTN, segmentation and reassembly are not required, because TCP is a connection oriented protocol and an RU of any length can be sent as a stream of bytes, (using a length field prefix to mark the record length, as described in Section 5.3.2, "MPTN_Hdr") without requiring any kind of "segmentation". However, when MPTN transport layer gateways are installed in APPN-to-TCP/IP network boundaries, RU segments created in a native SNA network must be reassembled somewhere in the session path before the application endpoint in an SNA-over-TCP/IP access node.

In TCP62, the reassembly is performed by the TCP62 code in the access node that receives the segments. In a native node with only a (full) APPC protocol stack, such reassembly is performed by a lower layer component of the stack (Path Control). However, in the MPTN framework for APPC over TCP/IP, the path control code is not involved in the session path) - because it belongs to the sub-layer-4 part of the APPC stack whose function is replaced by the TCP/IP stack - and therefore, reassembly must be performed by the TCP62 code.

When an MPTN gateway receives segmented data on the SNA network and sends it on the TCP/IP network, it adds the usual compensation headers to the data, and flows the TH subset and RH that is defined for all data over TCP. All LU6.2 segments that flow on a TCP connection consist of the length field and the compensation header. On all segments except the last one, the compensation header indicates that the data that follows is a record segmnent and not a full record. The last segment contains a regular record-over-stream compensation header, which is interpreted by the receiver as the last record to reassemble.

When an MPTN gateway sends segments over the TCP/IP network, only the first segment contains the TH subset and RH. Subsequent segments only contain the RU, making it easy for the TCP62 code on the reassembling node to concatenate the segments into a complete record. Note that this is different from the manner in which segments are sent over a native LU6.2 connection.

4.2. Pacing

Congestion on a TCP connection is indicated to a sockets application by blocking its send calls (or returning appropriate error codes to nonblocking send calls). For an LU6.2 connection set up using a TCP connection, congestion would be caused by the receiving end's transport provider not receiving data (due to buffer limitations on it's end coupled with the receiving application's behavior). The TCP62 code on the sending end should sense such congestion and give appropriate

feedback to the LU6.2 stack such that it can "fake out" the pacing protocols that the APPC stack in the sending node believes it is executing with the next hop node.

The SNA stack code on the sending node requests a pacing window by setting the pacing request bit in the RH. When the TCP62 code on the same node sees this bit, it should reset it before sending the packet over the TCP connection that represents the session (since the pacing request bit should not flow over the TCP connection), but it should remember that the sender had asked for a pacing window. If all SNA packets sent on the TCP connection, including the one whose pacing request bit was set initially, are accepted for transmission by the TCP stack (as indicated by the return code of the "send" socket call), then the TCP connection is not congested and an IPM can be sent to the SNA stack immediately. Otherwise, the TCP62 code should wait until all packets including the one requesting a pacing window is sent before sending the IPM.

On the receiving node, when TCP62 code receives an indication that there is data available on a TCP connection that represents a session, it checks if SNA buffers are available for receiving that data. It initially starts with one buffer for receiving an RU (this is true of native SNA sessions too). Once that buffer has been filled with inbound data, TCP62 should set the pacing request bit on the RU before giving it to the local SNA stack code. Depending on system buffer availability, the SNA stack will either send an IPM immediately, or when buffers become available. The TCP62 code should keep track of the IPM size to determine how many more inbound RUs can be received from the TCP connection before it has to ask for another window.

If there is a system buffer shortage in the receiving node, the TCP62 code will stop receiving IPMs from the local SNA stack, and will therefore stop receiving data on the TCP connection. When this happens, the TCP connection's receive queue will fill up, it will stop sending TCP window updates to the sender to reflect the local congestion, the sender (TCP62 code on the sending node) will see the congestion on it's "send" socket calls, and therefore, the TCP62 code will stop sending IPMs to the sender's SNA stack. That in turn will cause the SNA stack on the sending node to apply back pressure to the application sending data. This application of back pressure based on TCP flow control, and the virtualization of SNA hop-by-hop pacing within a node, is how pacing of LU6.2 connections is performed in TCP62.

5. TCP62 Formats

5.1. Migration Considerations

In order to ease migration for future functionality that may be added to the architecture, MPTN defines a mechanism that allows MPTN nodes to use new functions when their partners support it and still be able to interoperate with back-level nodes. Every MPTN command and optional field contains a processing specification field that tells the receiving node how to handle the command or optional field based on a number of criteria. All MPTN nodes must understand the processing specification so that they can take the correct action when they encounter unrecognized fields. If a node receives an MPTN command with an illegal command processing specification value, it will discard the command.

The processing specification is a one byte field in the common prefix that specifies:

- Whether the command is a request or a response.
- If a response, whether it is positive or negative.
- The actions that should be taken by an access node or MPTN Gateway that does not recognize the command type.
- If a negative response, the reason for the failure (rejected by the partner or not understood by the partner).

For a more detailed description of the bits in the processing specification field, please refer to Section 5.2.1, "Common Prefix".

5.1.1. Processing Rules

The following rules govern the behavior of both up-level and back-level nodes in order to make orderly migration possible:

1) Every optional field and every command contains a one-byte processing specification field that specifies the actions that should be taken by either an access node or gateway that does not recognize the command or field type.

2) A destination node that fails to recognize a command type or optional field tag uses the processing specification value to choose an action. The possible actions are detailed in Section 5.1.2, "Processing Actions".

3) Commands are processed before optional fields. If the command is valid, then the setting of the processing specification relies solely on the result of any optional field processing.

4) All optional fields should be processed, unless an error has already been encountered.

5) It is possible that future format changes may require multiple copies of the same optional field, while a back-level node may understand only a single instance of the field. If this happens, the receiver uses the first instance of the field and treats any additional copies of the field that it does not expect as unrecognized fields, i.e., it should ignore or fail them as specified in their processing specification fields.

6) Certain required fields and optional fields (those having an overall length count) may be extended in the future by adding additional information to the end of the existing format. This may be done if and only if the success of the command does not depend on the user being able to process the additional information. For example, the user transport requirements field may be extended with additional user characteristics that a back-level destination access node does not recognize and can ignore. In this case, a destination access node should not fail a command because the field is longer than expected. When the destination access node returns such a field in a reply, the following processing applies:

   a) When the destination access node recognizes the additional information it must echo it back.

   b) When the destination access node does not recognize the additional information, it is required that the destination access node drop the part not recognized. In this case, it cannot be assumed that if the additional information is returned, it is understood.

7) Reserved fields fields are set to zero by the sender and ignored by the receiver. This allows future use of these bits without breaking a back-level node.

5.1.2. Processing Actions

This section describes the actions taken by a destination node that does not recognize a field or command.

5.1.2.1. Command

If the processing specification indicates that the destination is not required to recognize the command, then the destination node should

discard the command.

If the processing specification indicates that the destination is required to recognize the command, then the destination node should:

- Set the following fields in the command processing specification:
  MPTN response indicator
  MPTN negative response indicator
  MPTN destination unrecognized indicator

- Set the Time To Live (TTL) field to initial value

- Optionally add diagnostics field

- Return the command to the source

5.1.2.2. Field

If the processing specification indicates that the destination is not required to recognize the field, then the destination node should set the MPTN destination unrecognized indicator in the optional field and proceed with the rest of the command processing.

If the processing specification indicates that the destination is required to recognize the field, then the destination node should:

- Set the following fields in the command processing specification:
  MPTN response indicator
  MPTN negative response indicator

- Set the following fields in the field processing specification:
  MPTN destination unrecognized indicator

- Set the Time To Live (TTL) field to initial value

- Optionally add diagnostics field

- Return the command to the source

5.2. Common Command Formats

Establishing, maintaining, and terminating LU6.2 connections over TCP/IP involves the use of several different command formats. These command formats do share some common characteristics. These common characteristics will be discussed in this section.

5.2.1. Common Prefix

Each command includes a 4-byte prefix, referred to in this draft as the "common prefix" field. This prefix identifies the command, provides processing information, and specifies the length of the command.

The format of the prefix is as follows:

Byte	Content
0	**Type indicator for this command.** Possible values are: X'80' MPTN_Connect X'83' MPTN_DG_OOB_Data X'84' MPTN_DG_KEEPALIVE_Hdr
1	**Flag field that indicates whether this is a request or response, and also provides processing information.**

Bit	Meaning
0	When set to 1, indicates that this is a response. Otherwise, this is a request.
1	Reserved in a request. When set to 1 in a response, indicates that this is a negative response.
2	Reserved in a request and a positive response. When set to 1, indicates that the command was recognized by the destination but rejected.
3	Reserved.
4	When set to 1 in a request, indicates that a gateway must reject the command if it is unrecognized by sending a negative response to the originator. When set to 0 in a request, indicates that a gateway must forward the unrecognized command to the destination. The bit is forwarded without modification in a response.
5	Reserved in a request. When set to 1 in a response, indicates that the gateway did not recognize this command. Bits 4 and 5 set to B'11' in a response indicate that the command failed because it was not recognized by

```
                        the gateway.

                    6   When set to 1 in a request, indicates that the
                        destination must reject the command if it is
                        unrecognized.  When set to 0 in a request, indicates
                        that the destination is to discard the unrecognized
                        command.

                        The bit is returned without modification in a
                        response.

                    7   Reserved in a request.  When set to 1 in a response,
                        indicates that the destination did not recognize this
                        command.

                        Bits 6 and 7 set to B'11' in a response indicate that
                        the command failed because it was not recognized by
                        the destination.
```

2 to 3 Length, in binary, of this command. Includes the four bytes used for this common prefix.

5.2.2. Optional Fields

Each command may contain one or more optional fields. These optional fields are preceded by a 4-byte prefix. This prefix identifies which optional field this is, provides processing information, and specifies the length of the optional field.

The format of this prefix is as follows:

```
Byte           Content

0              Type indicator for this optional field.  Possible values
               are:
               X'05' service mode
               X'0A' connection data
               X'18' user characteristics
               X'19' compensations required
               X'1A' optional compensations
               X'1C' node initialization ID
               X'28' sender connection alias
               X'29' receiver connection alias
               X'F0' diagnostics

1              Processing specification for this optional field.
```

Bit	Meaning
0 to 3	Reserved
4	When set to 1 in a request, indicates that the gateway must reject the command if this optional field is unrecognized. Echoed in a response.
5	Reserved in a request. When set to 1 in a response, indicates that a gateway did not recognize this optional field.
	Bits 4 and 5 set to B'11' in a response indicate that the command failed because this optional field was not recognized by a gateway.
6	When set to 1 in a request, indicates that the destination must reject the command if this optional field is unrecognized by the destination.
7	Reserved in a request. When set to 1 in a response, indicates that the destination did not recognize this optional field.
	Bits 6 and 7 set to B'11' in a response indicate that the command failed because this optional field was not recognized by the destination.

2 to 3 Length, in binary, of this optional field. Includes the four bytes used for used for this optional field prefix.

5.2.3. MPTN Addresses

Addresses used in MPTN command have a common format. Each MPTN address consists of an MPTN qualifier, an address mode, a node address, and a local address. These subfields are described in the sections that follow:

When an address used in an MPTN command is optional, it is preceded by the 4-byte prefix used for all optional fields, described in "Optional Fields".

5.2.3.1. MPTN Qualifier

An MPTN qualifier is a 1-byte hexadecimal code that identifies the "address family" to which the address belongs. For LU6.2 over TCP/IP,

this value is always X'0B' which means that this address represents a SNA network-qualified LU name. In this case, all address values will be encoded in EBCDIC.

5.2.3.2. Address Mode

The address mode used in an MPTN address is a 1-byte hexadecimal code that identifies the type of addressing used. For LU6.2 over TCP/IP, the only possible value is 'X01' which represents an individual address.

5.2.3.3. Node Address

The node address in an MPTN address is the network address by which the node can be located. This address has a variable length, and therefore includes a 1-byte length field. Since a null node address is not permitted, the node address length will take values from 4 to 18 bytes for LU6.2 over TCP/IP because the node address corresponds to a network-qualified LU name. The format of the node address is:

Byte	Content
0	**Length (n + 1), in binary, of the node address**
1 **to n**	Node address

5.2.3.4. Local Address

Local addresses are not used with LU6.2 over TCP/IP; therefore, this field will always have a value of X'01'. This value represents a null local address.

5.2.4. Diagnostic Values

Diagnostics can be sent on any negative response. When used, they provide information about the reason for the failure of the command.

A negative response can include more than one Diagnostics field. The subfields of the Diagnostics field include a diagnostics prefix as described in the "Optional Fields" section, a primary return code, a secondary return code, an error detector address, and error detector data.

5.2.4.1. Primary Return Code Values

The primary return code describes the primary reason for the failure of the command. The format of this field is as follows:

Byte	Content
0	**Reserved**
1	**1-byte hexadecimal code of the command type that was in error**
2 to 3	Primary return code. The following values are defined for this field:

X'0001'	Connection limits exceeded
X'0002'	User not found
X'0003'	User not reachable
X'0006'	Service mode not supported
X'0007'	Rejected by user
X'0008'	User not listening
X'0009'	Time to live counter expired
X'0014'	User characteristics mismatch
X'0015'	Compensation unrecognized
X'0016'	Compensation mismatch
X'0017'	Error in user data
X'001E'	Connection data unexpected
X'001F'	Connection data missing
X'0020'	Compensations required field missing
X'0029'	Error in destination address
X'002A'	Error in source address
X'002B'	Optional fields out of sequence
X'002C'	Error in correlator
X'002D'	Length error: field too long
X'002E'	Length error: field too short
X'002F'	Error in receiver connection alias
X'0032'	Format error
X'0033'	Invalid header in MPTN_DG_OOB_Data
X'003C'	Internal processing error

Primary return codes X'0001', X'0002', X'0003', X'0020', X'0029', X'002A', X'002C', X'002D', X'002E', X'0032', and X'003C' provide additional diagnostic information through use of a secondary return code. Secondary return codes are described in the next section.

5.2.4.2. Secondary Return Code Values

Three alternative uses and formats are defined for this field. Format 1

specifies an offset into the command where an error was detected. This value is given relative to the start of the common prefix field, assuming offset 0 for the first byte of that field. Format 2 describes the secondary reason for the failure, where the secondary return code is defined in the context of the primary return code. Format 3 identifies the optional field in error and the instance of the field if multiple instances are present in the request.

```
Byte          Content

0             Format of the Secondary Return Code.  The
              following values are defined for this field:
              X'00'   Format 1
              X'80'   Format 2
              X'C0'   Format 3

1             Reserved

2 to 3        Dependent on format:

              Format 1: Offset to error in command.

              Format 2: For primary return code = X'0001':

                        X'0001' Rejected by access node
                        X'0002' Rejected by gateway

                        For primary return code = X'0002':

                        X'0005' Address unknown
                        X'0006' Qualifier unknown

                        For primary return code = X'0003':

                        X'000A'  Net ID unreachable
                        X'000B'  Node address unrecognized
                        X'000C'  Node unreachable

                        For primary return code = X'0029'
                          or X'002A'

                        X'000E'  Unrecognized MPTN
                                 qualifier
                        X'000F'  Unrecognized address mode
                        X'0010'  Invalid address mode
                        X'0011'  Missing node address
                        X'0012'  Node address too long
                        X'0013'  Invalid node address
```

X'0014'	Invalid MPTN qualifier
X'0015'	Node address too short
X'0016'	Host address unexpected

For primary return code = X'002C'

X'0018'	Out of range
X'0019'	Duplicate

For primary return code = X'0032'

X'001B'	Error in fixed field

For primary return code = X'003C'

X'001E'	Temporary error
X'001F'	Permanent error

Format 3: For primary return code = X'0020', X'002D', or X'002E':

Byte	Meaning
0	Field identifier of the optional field in error.
1	Index (starting from 0) of the instance of the field having the error.

5.2.4.3. Error Detector Address

The error detector address contains the address of the node that detected the failure. It is an MPTN address, and is therefore structured according to the description in "MPTN Addresses". In this usage, the local address subfield is always present, but may or may not include an address value.

5.2.4.4. Error Detector Data

The node that detects the failure is allowed to send up to 254 bytes of additional information related to the failure. The format of this field is as follows:

Byte   Content

0	**Length (n + 1), in binary, of the error detector data.**
1 to n	Error data specified by the error detector. Format for this data is byte string.

5.3. Command Formats

5.3.1. MPTN_Connect

An MPTN_Connect request is the first command sent over a TCP/IP connection in order to establish a LU6.2 session. An MPTN_Connect response acknowledges that request, and indicates whether or not the connection was accepted. Both positive and negative responses are defined for the MPTN_Connect. The format of the command as it relates running LU6.2 applications over TCP/IP is as follows:

Field Name	Size Range (in bytes)	Contents
Record Length	4	Specifies the overall length of the command, including the four bytes used for its own specification.
Command Type	1	X'80'
Processing Specification	1	Bits 4 and 6 MUST be set to 1, indicating that the message must be recognized by both a gateway and the destination. The remaining bits in this field are set as described in the "Common Prefix" section.
Command Length	4	The size of this command minus the four byte record length field.
Time To Live	1	Used to prevent commands from circulating endlessly. When a command arrives with a value of 1, the command is either at its destination or, if not, will be rejected and returned to the initiator of the connection as a negative response.

Destination Address	5 to 512	Specifies the target address of the connection. This is the receiver of the MPTN_Connect request, and the sender of an MPTN_Connect response. This field is described in section "MPTN Addresses".
Source Address	5 to 512	Specifies the source address of the connection. This is the sender of the MPTN_Connect request, and the receiver of an MPTN_Connect response. This field is described in section "MPTN Addresses".
Correlator Suffix	2 to 9	Used to correlate a request with its response. The source address, taken together with the correlator suffix, uniquely identifies the MPTN connection for all time. Byte 0 is the length(n + 1), in binary, of the correlator suffix value. Bytes 1 to n represent the actual value.
User Transport Requirements		This section is composed of the six fields listed next.
User Transport Requirements Length	1	Length field for the user transport requirements. If the value in this field is greater than 15 (the length of the currently-defined subfields), all data beyond the first 15 bytes is ignored.
Maximum Record Length	4	Maximum size for a record that can be sent on this connection.
Maximum Expedited Data Length	4	Maximum size for an expedited data record that can be sent on this connection.
Maximum Termination Data Length	4	Maximum size for termination data that can be sent on this connection.
Termination Type	1	For LU6.2 over TCP/IP, this value is

		always X'20' - Duplex Abortive
Expedited Marking	1	For LU6.2 over TCP/IP, this value is always X'00' which specifies that the LU6.2 application does not need to mark the position in the normal data where expedited data was sent.

5.3.1.1. MPTN_Connect Optional Fields

The following fields are "MPTN Optional" fields that are required to run LU6.2 applications over TCP/IP. A description of each field is given and, if necessary, a format of subfields associated with the "MPTN Optional" field.

5.3.1.1.1. Service Mode

Specification of the level of transport service required in each transport network. Consists of the subfields Service Mode Prefix, MPTN Service Mode, and User-Defined Service Mode. The field is present when the sender requires a specific level of service.

Service Mode Prefix is an Optional Field prefix, as described in Section **5.2.2 "Optional Fields".** The type indicator (byte 0) is set to X'05'. Bits 4 and 6 of the processing specification (byte 1) are set to 1 and 0, respectively. This indicates that a negative response is required if the field is unrecognized by the destination. The remaining bits in the processing specification are set as described in Section 5.2.2 "Optional Fields".

MPTN Service Mode is a predefined service mode required by the user. The field is 1-byte in length with the following possible values:

X'00'	Only user-defined service-mode is specified. No MPTN-defined service mode will be used as a default.
X'01'	No specific service mode is required.
X'02'	High bandwidth is required.
X'03'	Fast response time is required.
X'04'	Secure service with high bandwidth is required.
X'05'	Secure service with fast response time is required.

For LU6.2 over TCP/IP, the service mode maps to the SNA class of service in the following way:

SNA Class of Service	MPTN Service Mode

CONNECT	X'01'
BATCH	X'02'
INTER	X'03'
BATCHSC	X'04'
INTERSC	X'05'
SNASVCMG	X'01'
CPSVCMG	X'01'

Note that at the time this draft was written, there were no known implementations of the secure modes, X'04' and X'05'. General security for TCP/IP is being defined by the IETF.

The User-Defined Service Mode identifies the service mode required by the transport user. When this field is supported by a gateway, then this value takes precedence over the value specified in MPTN Service Mode. The format of this field is as follows:

Byte	Content
0	**Length (n + 1), in binary, of the user-defined service mode**
1 to n	User-defined service mode (when present). Format is ASCII character string.

5.3.1.1.2. Connection Data

In a request, this field contains the BIND RU. In a positive response, this field contains a positive BIND_RSP. In a negative response, this field contains data related to the connection failure if any was specified; however, if any Diagnostics optional field except X'0007', Rejected by User, is present, this field echoes back the data sent on the MPTN_Connect request. Consists of the subfields Connection Data Prefix and Connection Data Value. Always present for LU6.2 over TCP/IP.

The Connection Data Prefix is an Optional Field prefix, as described in Section 5.2.2 "Optional Fields". The type indicator (byte 0) is set to X'0A'. Bits 4 and 6 of the processing specification Byte 1) are set to **0 and 1, respectively.** This indicates that a negative response is not required if the field is unrecognized by a gateway, but it is required if the field is unrecognized by the destination. The remaining bits in the processing specification are set as described in Section 5.2.2 "Optional Fields".

The Connection Data Value field contains the connection data specified by the user.

5.3.1.1.3. User Characteristics

Specifies the user characteristics for which support is requested on this connection. When more than one user characteristic is required, the user characteristic identifiers can be specified in any order. The contents of this field are not changed by a gateway. This field consists of the subfields User Characteristics Prefix and User Characteristics Value. The field is present when specific user characteristics are requested.

User Characteristics Prefix is an Optional Field prefix, as described in Section 5.2.2, "Optional Fields". The type indicator (byte 0) is set to X'18'. Bits 4 and 6 of the processing specification (byte 1) are both set to to 0, indicating that a negative response is not required if the field is unrecognized either by a gateway or by the destination. The remaining bits in the processing specification are set as described in Section 5.2.2, "Optional Fields".

User Characteristics Value is a list of user characteristic values representing the user characteristics to be used on this connection. The list consists of 1-byte unsigned binary values. The number of elements in the list is determined by subtracting 4 from the value stored in the length subfield (bytes 2 and 3) of the User Characteristics Prefix field. The value that can appear in this field for LU6.2 over TCP/IP is:

X'01'	Session Outage Notification (SON)

If the User Characteristics field is understood by the destination but a certain value is not supported then that value should be set to X'00' in the response.

5.3.1.1.4. Compensations Required

Specifies the compensations that will be used on this connection. When more than one compensation is required, the compensation identifiers can be specified in any order. The contents of this field can change at every gateway. This field consists of the subfields Compensations Prefix and Compensations Value. The fix is present when compensations are required.

The Compensations Prefix is an Optional Field prefix, as described in Section 5.2.2, "Optional Fields". The type indicator (byte 0) is set to X'19'. Bits 4 and 6 of the processing specification (byte 1) are both set to 1, indicating that a negative response is required if the field is unrecognized either by a gateway or by the destination. The remaining bits in the processing specification are set as described in

Section 5.2.2, "Optional Fields".

The Compensations Value field is a list of MPTN headers and command identifier values representing the compensations to be used on this MPTN segment. This field is a list of 1-byte unsigned binary values. The number of elements in the list is determined by subtracting 4 from the value stored in the length subfield (bytes 2 and 3) of the Compensations Prefix field. For LU6.2 over TCP/IP, the compensations required include:

X'00'	Record boundary marker
X'01'	Expedited message
X'03'	Expedited message acknowledgement
X'10'	Duplex-abortive termination
X'20'	Segmented Message
X'83'	MPTN_DG_OOB_Data message needed for expedited data

5.3.1.1.5. Optional Compensations

Specifies the optional compensations that are requested on this connection. When more than one optional compensation is specified, the compensation identifiers can be specified in any order. The contents of this field can change at every gateway. This field consists of the subfields Optional Compensations Prefix and Optional Compensations Value. This field is present when optional compensations are requested.

The Optional Compensations Prefix field is an Optional Field prefix, as described in Section 5.2.2, "Optional Fields". The type indicator (byte 0) is set to X'1A'. Bits 4 and 6 of the processing specification (byte 1) are both set to 0, indicating that a negative response is not required if the field is unrecognized either by a gateway or by the destination. The remaining bits in the processing specification are set as described in Section 5.2.2, "Optional Fields".

The Optional Compensations Value field is a list of MPTN headers and command identifier values representing the optional compensations to be used on this MPTN segment. This field is a list of 1-byte unsigned binary values. The number of elements in the list is determined by subtracting 4 from the value stored in the length subfield (bytes 2 and 3) of the Optional Compensations Prefix field. For LU6.2 over TCP/IP, the optional compensations include:

X'84'	MPTN_DG_KEEPALIVE_Hdr message needed for session outage notification

If the Optional Compensations field is understood by the destination but a certain value is not supported then that value should be set to X'00'

in the response.

5.3.1.1.6. Node Initialization ID

This field represents a value that uniquely identifies the instance of the sending node, e.g., a time corresponding to when the node was initialized. This field consists of a Node Initialization ID Prefix, and Node Initialization ID Value. This field is optional and used when there is a danger that a node may be terminated and reinitialized in less time than the keepalive process will detect.

The Node Initialization ID Prefix is an Optional Field prefix, as described in Section 5.2.2, "Optional Fields". The type indicator (byte 0) is set to X'1C'. Bits 4 and 6 of the processing specification are both set to 0, indicating that a negative response is not required if the field is unrecognized either by a gateway or by the destination. The remaining bits in the processing specification are set as described in Section 5.2.2, "Optional Fields".

The Node Initialization ID Value field specifies the node initialization ID, e.g., a time stamp. The field may be from 2 to 8 bytes in length.
sp

5.3.1.1.7. Diagnostics

This field is only allowed on negative MPTN_Connect responses. For a description, please refer to Section 5.2.4, "Diagnostic Values".

5.3.2. MPTN_Hdr

MPTH headers are 1-byte fields that are inserted before the user's data on an MPTN connection. The field specifies which compensation, if any, is in use. The possible identifiers required for running LU6.2 applications over TCP/IP include:

X'00'	Record boundary marker
X'01'	Expedited message
X'03'	Expedited message acknowledgement
X'10'	Duplex-abortive termination
X'20'	Segmented message

User data may follow the MPTN header. When user data follows MPTN header X'10', then that data is an UNBIND specified by the transport user. User data is present when the transport user sends data associated with the MPTN header. The user data field is not required for some compensations, such as the various termination compensations

when they are used without user termination data and acknowledgements.

For LU6.2 over TCPI/IP, the MPTN header will be preceded by a 4-byte record length that specifies the overall length of the message including the four bytes used for its own specification.

5.3.3. MPTN_DG_OOB_Data

The MPTN_DG_OOB_Data is a command defined for sending data on a separate path from the connection ("out of band"), to ensure that it reaches the partner. This is command can be used, for example, to insure that SNA expedited data reaches the target node. The format of the command as it relates to running LU6.2 applications over TCP/IP is as follows:

Field Name	Size Range (in bytes)	Contents
Record Length	4	Specifies the overall length of the command, including the four bytes used for its own specification.
Command Type	1	X'83'
Processing Specification	1	Bits 4 and 6 MUST be set to 0 and 1, respectively. This indicates that a negative response is not required if the message is unrecognized by a gateway, but it is required if the message is unrecognized by the destination. The remaining bits in in this field are set as described in Section 5.2.1, "Common Prefix".
Command Length	4	Represents the size of this entire message. It includes the four bytes used for the Common Prefix, but does not include the four bytes for the Record Length field, the size of the MPTN Header, or the size of the User Data.
Connection Sequence Number	2	Sequence number used to correlate this datagram with activity on a specific connection.
Correlator Address	5 to 512	Specifies the address of the user that

		initiated the connection that the out-of-band data relates to. This field is described in Section 5.2.3, "MPTN Addresses".
Correlator Suffix	2 to 9	Used with the Correlator Address field to correlate this out-of-band data to its connection. (This is the same value that was sent in the MPTN_Connect command for the connection.) Byte 0 is the length (n + 1), in binary of the correlator suffix value. Bytes 1 to n represent the actual value.
Sender Connection Alias	9 to 516	Specifies a connection alias that, when used in the acknowledgement, will enable the sender to quickly identify the associated connection. Consists of the following subfields: Sender Alias Prefix, Sender Alias MPTN Qualifier, Sender Alias Address Mode, Sender Alias Node Address, and Sender Alias Local Address. Sender Alias Prefix is an Optional Field prefix, as described in Section 5.2.2, "Optional Fields". The type indicator (byte 0) is set to X'28'. Bits 4 and 6 of the processing specification (byte 1) are both set to 0. This indicates that a negative response is not required if the field is unrecognized by a gateway or by the destination. The remaining bits in the processing specifications are set as described in Section 5.2.2, "Optional Fields". The remaining subfields are described in Section 5.2.3, "MPTN Addresses". Note that the Sender Alias MPTN Qualifier must be set to X'7F' and the Sender Alias Address Mode must be set to the value X'01'. This field is present when the sender (of a request or response) wants to communicate a local form address to the destination and is not required.
Receiver	9 to 516	Specifies a connection alias that

```
  Connection Alias                enables the receiver to quickly
                                  identify the connection for which this
                                  out-of-band data was sent.  This is the
                                  connection alias that the sender
                                  received earlier from the partner.
                                  Consists of the following subfields:
                                  Receiver Alias Prefix, Receiver Alias
                                  MPTN Qualifier, Receiver Alias Address
                                  Mode, Receiver Alias Node Address, and
                                  Receiver Alias Local Address.  Receiver
                                  Alias Prefix is an Optional Field
                                  prefix as described in Section 5.2.2,
                                  "Optional Fields". The type indicator
                                  (byte 0) is set to X'29'.  Bits 4 and 6
                                  of the processing specification (byte
                                  1) are both set to 0.  This indicates
                                  that a negative response is not
                                  required if the field is unrecognized
                                  by a gateway or by the destination.
                                  The remaining bits in the processing
                                  specification are set as described in
                                  Section 5.2.2, "Optional Fields".  The
                                  remaining subfields are described in
                                  Section 5.2.3, "MPTN Addresses".  Note
                                  that the Receiver Alias MPTN Qualifier
                                  MUST be set to X'7F' and the Receiver
                                  Alias Address Mode MUST be set to the
                                  value X'01'.

                                  This field is present when the sender
                                  has previously received a sender
                                  connection alias from the partner.

Maximum Datagram   4              This is an Optional Field  prefix as
  Common Prefix                   described in Section 5.2.2, "Optional
                                  Fields".  The type indicator (byte 0)
                                  is set to X'2E'.  Bits 4 and 6 of the
                                  processing specification (byte 1) are
                                  both set to 0.  The remaining bits in
                                  the processing specification are set
                                  as described in Section 5.2.2,
                                  "Optional Fields".

                                  This field is present when TCP/IP
                                  stack can receive a datagram bigger
                                  than the largest datagram
                                  guaranteed to be receivable by TCP/IP.
```

Maximum Datagram Value	4	This field represents the maximum datagram receive size supported by the sending TCP/IP stack. This field is only present when the Maximum Datagram Common Prefix is present. The value is expressed in a binary form and is limited to 4 bytes in length.
Diagnostics	18 to 779	This field is only allowed on negative MPTN_DG_OOB_Data responses. For a description, please refer to Section 5.2.4, "Diagnostic Values".
MPTN Header	1	Indicates the compensations associated with this command. This field MUST be in a request and MUST NOT be in a response. The values are expressed in a hexadecimal format. The following values are allowed: X'01' user expedited data X'10' duplex-abortive termination
User Data	0 to (2 32 - 32)	Contains the transport user's data. If MPTN Header is X'01', then this is user expedited data. If MPTN header is X'10', then this is termination data specified by the transport user. This field may be present in a request depending on the MPTN Header. This field MUST NOT be present in a response.

5.3.4. MPTN_DG_KEEPALIVE_Hdr

The MPTN_DG_KEEPALIVE_Hdr is a command defined for detecting session outage notification in a timely manner.

Field Name	Size Range (in bytes)	Contents
Record Length	4	Specifies the overall length of the command, including the four bytes used for its own specification.
Command Type	1	X'84'

Processing Specification	1	Bits 4 and 6 are both set to 0, indicating that no negative response is required if the command is unrecognized by a gateway or the destination. The remaining bits in the processing specification are set as described in Section 5.2.2, "Optional Fields".
Command Length	2	This value represents the size of this entire command excluding the four byte record length field.
Source Address	5 to 512	Specifies the provider address of the sender of the keepalive command. This is the sender of an MPTN_DG_KEEPALIVE_Hdr request, and the receiver of the MPTN_DG_KEEPALIVE_Hdr response. The contents of this field are described in Section 5.2.3, "MPTN Addresses".
Destination Address	5 to 512	Specifies the provider address of the receiver of the keepalive command. This is the receiver of an MPTN_DG_KEEPALIVE_Hdr request, and the sender of the MPTN_DG_KEEPALIVE_Hdr response. The contents of this field are described in Section 5.2.3, "MPTN Addresses".
Diagnostics	18 to 779	This field is only allowed on negative MPTN_DG_OOB_Data responses. For a description, please refer to Section 5.2.4, "Diagnostic Values".

SECURITY

This draft does not attempt to address the issue of security. This draft does not affect any existing level of SNA security that exists today.

REFERENCES

[1] X/OPEN CAE Specification, Multiprotocol Transport Networking (XMPTN): Access Node (ISBN: 1-85912-106-3, P521)

[2] X/OPEN CAE Specification, Multiprotocol Transport Networking (XMPTN): Data Formats (ISBN: 1-85912-111-X, C522)

[3] Multiprotocol Transport Networking (MPTN) Architecture: Technical Overview, IBM Document #GC31-7073.

AUTHORS' ADDRESSES

James Carmichael
IBM Corporation
P.O. Box 12195
Research Triangle Park, NC 27709
USA

phone: +1 919-254-5798
email: carmich@us.ibm.com

Soumitra (Ronnie) Sarkar
IBM Corporation
P.O. Box 12195
Research Triangle Park, NC 27709
USA

phone: +1 919-254-6461
email: sarkar@us.ibm.com