INTERNET-DRAFT
Expires: February 28, 1998

D.Piscitello,
L.Phifer
Core Competence
Y.Wang,
R.Hovey
Bellcore
August 28, 1997

Mobile Network Computing Protocol (MNCP)
<draft-piscitello-mncp-00.txt>

Status of this Memo

Abstract

This memo documents an architecture and protocol for mobile network computing. The protocol described herein provides session control and reliable delivery services to accommodate mobile client access to internetworked applications within a 'client-agent-server' architecture. Client middleware based on this architecture can operate over wireless data networking services such as RAM, ARDIS, CDPD, PCS data services and wireless local area networks. This client middleware can be implemented using any standard application programming interface to a commercial UDP/IP stack on Hand-held PC's (HPC), personal digital assistants (PDA's), four-line browsers or 'smart' phones as well as laptop and desktop computers.

Table of Contents

1. Introduction

1.1 Motivation

Mobile network computing, if constrained by consumer interest alone, would at this point in time increase more rapidly than the growth of the Internet itself. Applications that drive consumer interest -- access to the public web and intranets, remote access to corporate and public databases, unified messaging and two-way paging -- are already present and widely available, having already been enabled by public and enterprise IP-based internetworking.

The need to access internetworked applications remotely has already been established, but the means of addressing that need are only partially satisfied through the use of wireline and portable (laptop) PC solutions.

The rapid acceptance of cellular telephony provides a strong indication of how quickly similarly untethered computer communications will be embraced by consumers. Recent technology advances now make it possible to produce handheld devices that are as small as cellular phones yet smart as portable PC's. These devices are very adapted for wireless

communications environments, better able to maintain signal strength and intelligently manage power consumption. It is thus likely that mobile network computers (MNCs), hand-held PC's (HPCs), personal digital assistants, and four-line browser or "smart" phones operating over wireless data networks will complement (or inherit) existing remote access alternatives, and create a potentially enormous consumer market for wireless data networking services such as RAM, ARDIS, CDPD, and PCS data services.

This new class of mobile computing devices (MCD's) will often operate in low bandwidth, high latency environments, where it is important to minimize communications consumption. Such environments are not, however, the exclusive operating domains for every mobile computing device, and a device does not have to be among those types previously enumerated to be mobile. Any classification of MCD's must also include desktop and docking laptop computers in wireless LAN environments, where mobility within a building or campus is provided by wireless Ethernet or similar technology. It is also true that many mobile computing devices may be used in both wireless and wireline environments: change a network interface card (NIC) on many of these devices, and the MCD can operate over analog dial, ISDN, or in a LAN.

1.2 Design Goals

Because of the diverse nature of Mobile Computing Devices, the communications environments over which they may operate, and the applications MCD's may provide, several design goals emerge.

1) It is important to minimize communications consumption when low bandwidth, high latency networks are used by MCDs;

2) Applications should operate well irrespective of wireless or wireline network characteristics; and

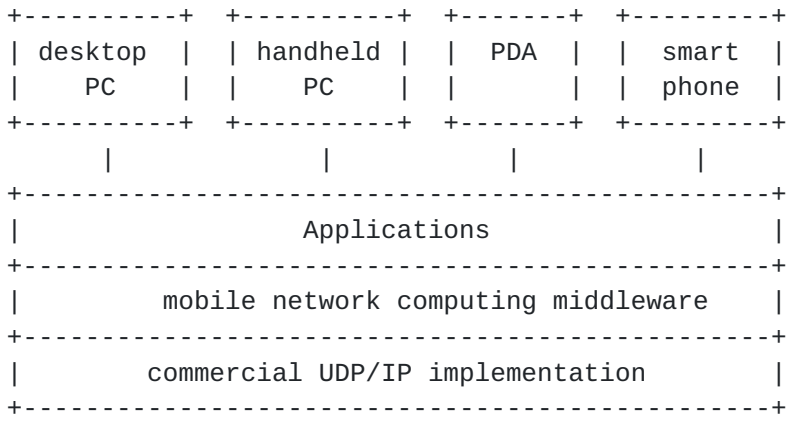
3) It should be possible for certain classes of MCD's to operate in a disconnected state.

4) It should be relatively simple for end users to change communications environments; optimally, an end user would be able to install the appropriate network interface and begin communicating over a different type of network.

5) It is desirable to have a mechanism to allow subscriber identification and authentication to be IP address independent.

6) It is desirable to minimize communications consumption for low bandwidth devices with limited battery life.

7) Both mobile computing devices and mobility servers should be able to initiate communications and transfers of data; i.e., client initiated or pull" applications as well as server initiated or "push" applications should be accommodated.



Mobile client applications will operate on wireless networks in a bandwidth-latency range where many commercial TCP's have insufficient tuning parameters to permit efficient operation. Custom TCP's might be developed to accommodate the specific bandwidth-delay characteristics of wireless networks, but these custom TCP's would need to be installed in all networked hosts with which the user wishes to communicate, which is not practical.

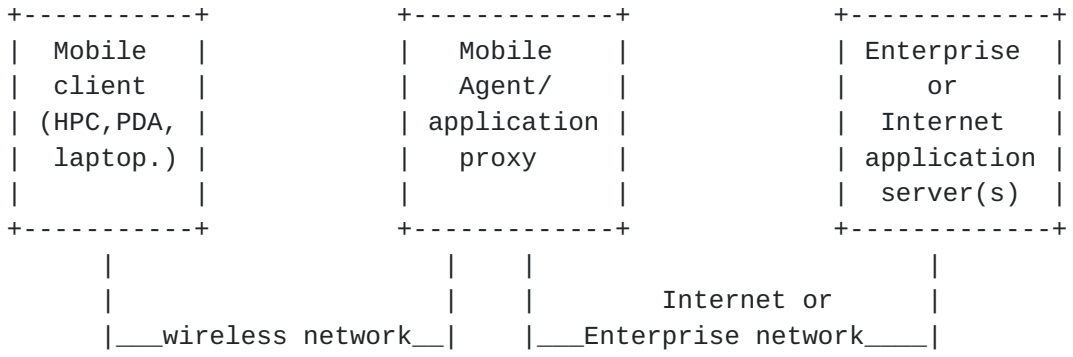
To meet the design goals enumerated, and to avoid situations where the end user would be responsible for reconfiguring TCP for each environment, or where the user might have to install a different TCP entirely to operate over a LAN or wireless WAN, we believe it is appropriate to build a middleware element that can operate on top of any commercial, off-the-shelf UDP/IP implementation.

[Note: It is conceivable that a standard TCP could be adapted to satisfy the network transparency design goals. The difficulty with this approach is that it will be some time before this can propagate into existing TCP implementations. More importantly, the existing TCP

architecture does not allow optimizations for minimizing communications consumption for low bandwidth devices with limited battery life, as will be described with MNCP.]

These requirements suggest that important efficiencies can be achieved by adopting an agent-enabled, transmission independent messaging paradigm. This client-agent-server architecture allows for the introduction of increased efficiencies such as session level data compression.

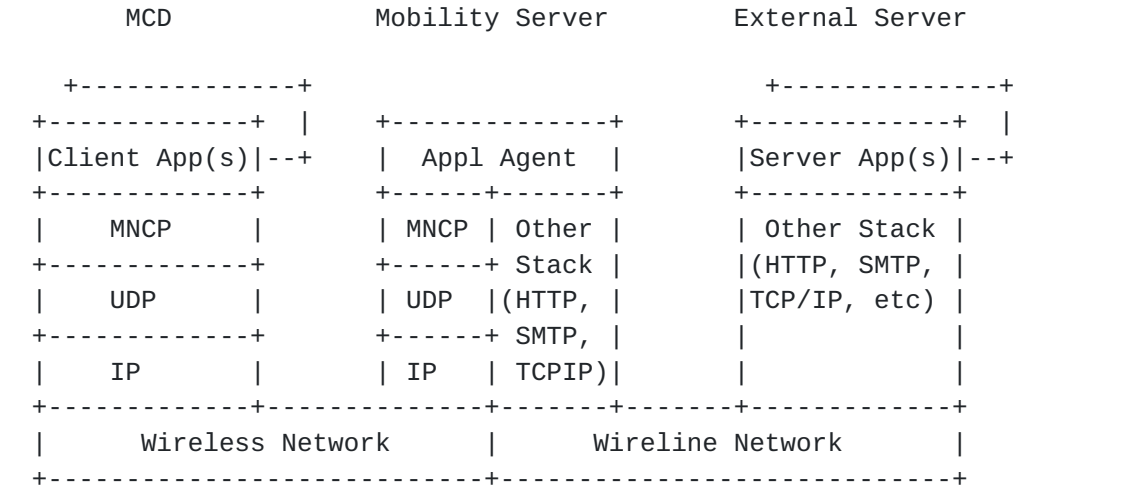
[Note: This client-agent-server relationship is euphemistically referred to as a thin-client architecture. However, it is appropriate to consider so-called thin clients as one of many, rather than the only, type of client accommodated by the MNC architecture.]



In addition, client-server behavior and the "chatty" protocol behavior associated with client-server (web) interaction and transactions can be optimized by introducing a degree of parallelism, i.e., by adopting common service or "session layer" framing as well as application specific framing, on top of traditional transfer control framing. In addition to the operational efficiencies introduced with this approach, mechanisms for providing reliable delivery over wireless technologies can be developed and applied in application, rather than TCP "kernel" and operating system, space.

1.3 Mobile Network Computing Architecture

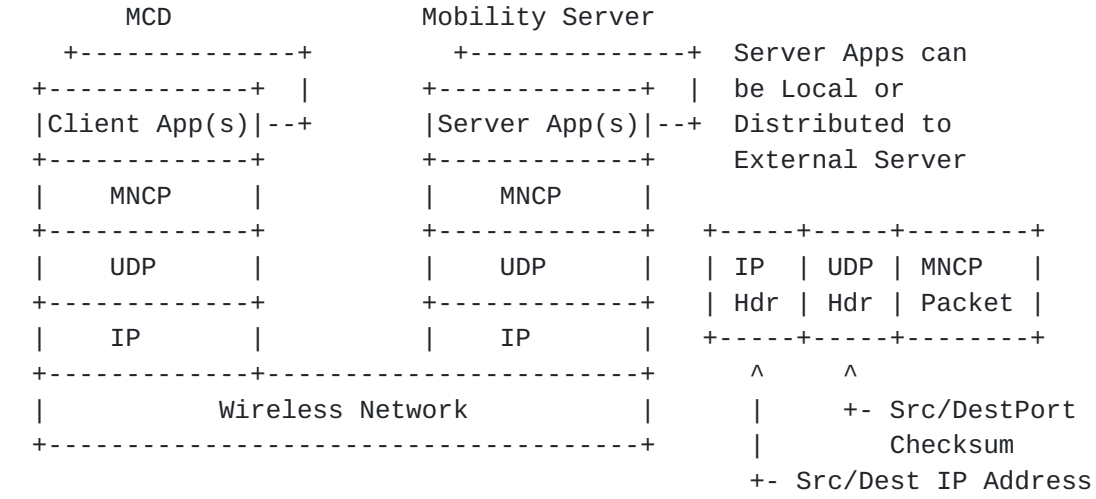
The Mobile Network Computing architecture consists of a middleware service component to support user registration and authentication, data transfer (with compression) and reliable delivery. A diverse set of application service components may ride on top of this middleware, each providing application-specific services such as mobile (unified) messaging, paging, browsing, and remote database access.



This internet-draft describes the Mobile Network Computing Protocol, which provides the services ascribed to the middleware component of the architecture.

1.4 Relationship of MNCP to other Internet Protocols

MNCP is designed to be implemented on top of the Datagram protocol (UDP). MNCP packets have an IP header, a UDP header, and an MNCP header.



The source and destination address fields of the IP header are used by MNCP to identify the requesting and responding hosts. For example, a request initiated by an MCD to a Mobility Server will carry the MCD's IP address in the source field and the Mobility Server's IP address in the destination field.

The source and destination port fields of the UDP header are used by identify the requesting and responding MNCP's. An MNCP implementation listens to an assigned "well known" UDP port number (to be assigned and recorded by IANA[2]) for incoming requests, and demultiplexes them to the appropriate application based upon Service ID (see Section 4.5.2). For example, a request initiated by a mobile messaging client application will carry the application's UDP port number in the source port field (selected from the range of values for UDP "ephemeral" or

client ports) and the "well known" port number for MNCP in the destination port field.

The UDP header length field reflects the total size of the MNCP packet. MNCP relies upon the UDP header's checksum field and error checking to protect the MNCP packet. MNCP provides end-to-end acknowledgment, retransmission, flow control, and segmentation as needed to insulate supported applications from the diverse characteristics of the underlying mobile network.

Client and server applications supported by MNCP are identified by a Service ID field carried in the first MNCP packet of each packet sequence. In order to send or receive MNCP packets with a given Service ID, the MCD must register for that service with the Mobility Server. The MNCP is responsible for authenticating the client and performing access control, based upon Subscriber ID and Password fields supplied by the MCD. The MNCP relays messages between server applications and client applications currently registered for that Service ID.

1.5 Requirements

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC 2119[5].

1.6 Terms

This memo uses a number of terms to describe components of the MNCP. Other common terms are used as specified in RFC 1983[4].

Mobile Computing Device (MCD)
PDA, laptop, hand-held device, desktop PC, or other network computer connected via wireless technology to a Mobility Server.

Mobility Server (MS)
The system which acts as an application layer gateway or agent between

MCDs and networked application services such as mobile messaging or web-enabled database access.

Client Application
An application located on an MCD which uses MNCP to communicate with Server Applications.

Server Application
An application conceptually located on a Mobility Server, but which may be physically distributed (i.e., a service-specific application gateway on the Mobility Server relays messages to and from a server application running on an external server).

Packet
The basic unit of MCNP communication, consisting of a structured sequence of octets matching the syntax defined in Sections 3-4 and transmitted over wireless networks connecting an MCD and its Mobility Server.

2. Protocol Overview

This section provides a brief overview of each type of service, enumerating the features provided by each service.

MNCP operates over the User Datagram Protocol, and relies on UDP protocol for protocol demultiplexing and data integrity . MNCP provides the following basic services:

- A single packet (acknowledged datagram) reliable delivery service supports applications where a single datagram request and reply sequence is sufficient for application data transfer.

- A multi-packet reliable delivery service supports applications requiring the reliable delivery of arbitrarily long data.

- A set of registration and request/response correlation (message flow support) services collectively referred to as SESSION CONTROL.

The MNC protocol consists of a message set of four basic packet types: COMMAND, NOTIFICATION, DATA, and ACKNOWLEDGE. Reliable delivery and session control services are provided through the use of Information Elements (IE's) encoded in these four packet types. The encodings of Information Elements for each packet type are described in sections 3 and 4.

The remainder of section 2 provides an overview of how MNCP provides data transfer and session control services to mobile applications.

2.1 Single Packet Reliable Delivery Service

The single packet reliable delivery service is an acknowledged datagram service. The service makes use of the COMMAND (PT_CMD) and ACKNOWLEDGE (PT_ACK) packets. The basic features of the service are:

- Data transfer
- Packet correlation
- error detection and recovery using positive acknowledgment with retransmission based on timeout

The single packet reliable delivery service is used when an application invokes the service with a request to send data, and a single MNCP datagram request and reply sequence is sufficient for application data transfer. This is true when the total amount of data to be sent is less than or equal to the default packet size (470 octets).

The MNCP constructs a COMMAND packet as follows. Common header fields are populated as discussed in section 3.2. A correlation identifier value is chosen for the packet exchange by the MNCP and encoded in the COMMAND packet (see section 2.3.5). Session Control information supplied by the application in the MNCP service request (service, function, and subscriber identifiers, and the subscriber password, see Section 2.3) are encoded in the packet header, along with any application-specific information elements supplied in the request. Application data are then appended to the header as "payload".

The COMMAND packet is sent in a UDP packet to the well-known MNCP port (source UDP port value is assigned from UDP client space), and a retry timer is initiated by the sender. If an acknowledgment is not received before the retry timer expires, the COMMAND packet is resent. If a specified maximum number of retry attempts is exceeded and an ACKNOWLEDGMENT is not received, the failure is reported to the application. If an ACKNOWLEDGMENT is received, a confirmation (delivery success or failure) is returned to the application.

When a COMMAND packet is received, the MNCP parses the packet and composes and returns a ACKNOWLEDGMENT packet containing a negative ACK code if errors were detected. Otherwise, the MNCP forwards the packet to the MNCP session control for processing (see section 2.3). If subscriber authentication and application service access control are successful, session control passes the contents of the data payload to

the application service indicted in the header, and returns a positive acknowledgment to MNCP. If authentication fails, or the subscriber is not authorized to use this application, session control will return a negative acknowledgment code to MNCP. Upon reception of a response from session control, the MNCP composes and returns an ACKNOWLEDGMENT packet with a positive or negative ACK code (the ACK code may be negative, indicating a session control failure, such as an invalid subscriber identifier or password).

2.2 Multi-Packet Reliable Delivery Service

The multi-packet reliable delivery service is used when an application attempts to send a message that is longer than the default packet size offered (see section 3.4), i.e., when the entire user data, uncompressed, cannot be transferred in a single packet. The service makes use of the NOTIFICATION (PT_NTFN) DATA (PT_DATA) and ACKNOWLEDGE (PT_ACK) packets. The basic features of the service are:

- Data Transfer
- Packet correlation
- Packet size selection
- Data compression method selection
- Error detection and recovery using positive acknowledgment with retransmission based on timeout
- Flow control
- Segmentation and reassembly
- Data compression (when selected)

2.2.1 NOTIFICATION Packet Processing (Sender)

When the multi-packet reliable delivery service is used, the MNCP constructs a NOTIFICATION packet to initiate a sequence of data packets. The purpose of the NOTIFICATION packet is to convey to the receiver certain information regarding overall and compressed size of the data to be transferred, and to propose a data compression method and maximum packet size to use when transferring subsequent data in the context of this multi-packet transfer.

The NOTIFICATION packet is always default packet size (470 octets) or less, and is constructed as follows. Common header fields are populated as discussed in section 3.2. A correlation identifier value is chosen

for the packet exchange by the MNCP (see section 2.3.5) and encoded in the NOTIFICATION packet. The sequence number in the NOTIFICATION is set to an initial value of zero (all subsequent DATA packets within the same packet sequence are incremented sequentially by one). The total (uncompressed) length of the application message to be sent is encoded

in the packet as the Original Message Length.

To increase network efficiency, the sender can propose to use a packet size larger than the default packet size. For this version of the protocol, the maximum packet size that can be proposed is 2048 octets (see section 3.4). The sender can also propose to use data compression, by specifying a compression method in the Data Compression Option. Proposing a larger packet size and a data compression method are options in the MNCP.

Session Control information supplied by the application in the MNCP service request (service, function, and subscriber identifiers, and the subscriber password, see Section 2.3) are encoded in the packet header, along with any application-specific information elements supplied in the request.

The NOTIFICATION packet is sent in a UDP packet to the well-known MNCP port (source UDP port value is assigned from client space), and a retry timer is initiated by the sender. If an ACKNOWLEDGMENT packet is not received before the retry timer expires, the NOTIFICATION packet is resent. If a specified maximum number of retry attempts is exceeded and an ACKNOWLEDGMENT packet is not received, a failure is reported to the application (see section 5.6).

If an ACKNOWLEDGMENT packet containing a positive ACK code is received, the sender begins transferring application data (see section 2.2.3). If the receiver has accepted an increased packet size, then the sender extracts the packet size specified in the ACKNOWLEDGMENT packet and uses this for subsequent message transfer. The packet size indicated in the ACKNOWLEDGMENT packet may be equal to or less than the size proposed by the sender. If the sender has proposed a data compression method, a positive ACK code indicates that the receiver has agreed to the data compression option proposed by the sender in the NOTIFICATION packet being ACKed.

The receiver may return a negative code to reject the compression method proposed in a NOTIFICATION packet, and may propose an alternative compression methods in the ACKNOWLEDGEMENT packet, to expedite the compression selection process. If the sender supports the data compression method proposed, the sender resends the NOTIFICATION, identifying the data compression proposed by the receiver in the ACKNOWLEDGMENT packet; alternatively, the sender may bid a new method. The receiver may again return a negative acknowledgment code to reject the proposed compression method, and (optionally) propose an alternative method. This form of "bidding" continues until a mutually acceptable compression method is identified (no compression is a legitimate option). The sender recognizes that compression selection has concluded when it receives an ACKNOWLEDGMENT packet containing a positive acknowledgment code.

[NOTE: Under this compression selection scheme, the sending MNCP must compress the data using a particular algorithm before it sends the PT_NTFN in order to convey the compressed length. This reflects current implementation practice. This memo does not preclude the use and future specification of stream compression algorithms that could be more closely coupled with an underlying transmission service, to optimize performance.]

2.2.2 NOTIFICATION Packet Processing (Receiver)

The MNCP parses an incoming NOTIFICATION packet and composes and returns a ACKNOWLEDGMENT packet containing a negative ACK code if errors were detected in the common header. Otherwise, the receiver examines the NOTIFICATION packet to determine if the sender has proposed any options. If the sender has proposed a packet size greater than the default packet size, the receiver may agree to use the larger packet size or it may propose an alternative size that is less than the size specified in the NOTIFICATION packet. The receiver can propose a smaller packet size and still return a positive acknowledgment.

If the NOTIFICATION proposes a data compression method that is not supported by the receiver, the receiver may reject the proposed data compression method and propose an alternative method in the ACKNOWLEDGMENT packet returned to the sender. As described in section 2.2.1, a form of "bidding" continues until both receiver and sender identify a mutually acceptable compression method.

When processing associated with multi-packet reliable delivery is completed by the receiver, the MNCP forwards the NOTIFICATION packet to the MNCP session control for processing (see section 2.3). Upon reception of a response from session control, the MNCP composes and returns an ACKNOWLEDGMENT packet with an ACK code indicating success or failure of session control processing.

2.2.3 Data Transfer (Sender)

When the NOTIFICATION processing is completed, the MNCP attempts to transfer data. The original application data submitted in the request is first compressed (if compression is selected), then segmented into a sequence of DATA packets containing data payloads of size {negotiated packet size - DATA packet header information}, i.e., when constructing a DATA packet, the MNCP attempts to create "n" packets of this fixed size. The last segment of application data transferred in this sequence of DATA packets may contain fewer octets than the negotiated packet size minus the header.

The processing and transmission of a sequence of DATA packets is as

follows. All packets in a sequence of DATA packets carry the same Correlation Identifier as the NOTIFICATION packet. A Data Offset is encoded in each DATA packet to assist in the reassembly of the application message. The first in a sequence of DATA packets has a Data Offset value of zero . When compression is used, the sending MNCP will fill the first DATA packet to the maximum payload available with compressed data, otherwise, each packet is filled to the maximum payload available with a segment of the original uncompressed message.

A sequence number is encoded in each DATA packet to assist in determining packet order. The sequence number of the initial DATA packet in a sequence is set to one (1).

For each additional DATA packet in the sequence, the Data Offset value represents the offset from the beginning of the transmitted data (hence, if the message was compressed before it was sent, the offset is relative to the beginning of the compressed message, not the original uncompressed message). The sequence number value is incremented by one for each subsequent segment created. The sequence number is not altered if a packet is retransmitted.

Each packet in a sequence of DATA packets except the final DATA packet carries an indication that more DATA packets follow this packet. The final DATA packet may contain less that the maximum data payload number of octets, and must not be padded.

Each DATA packet is sent as a UDP packet to the source port which sent the last ACKNOWLEDGMENT packet, and a retry timer is initiated by the sender. If an acknowledgment is not received before the retry timer expires, the DATA packet is resent. If a specified maximum number of retry attempts is exceeded and an ACKNOWLEDGMENT is not received, the failure is reported to the application (see section 5.6). If an ACKNOWLEDGMENT containing the sequence number of the DATA packet sent is received, the next DATA packet in the sequence is transmitted (out of sequence ACKNOWLEDGMENT packets are discarded). Upon reception of a positive ACKNOWLEDGMENT to the final DATA packet, a confirmation is provided to the sending application, indicating successful delivery.

2.2.4 Data Transfer (Receiver)

When the receiver sends a positive ACKNOWLDEGMENT in response to a NOTIFICATION request, the receiving MNCP awaits the arrival of DATA packets.

Reliable delivery of data is achieved through the use of a stop-and-go with timeout retransmission mechanism. Each DATA packet must be

individually acknowledged by the receiving MNCP. The ACKNOWLEDGMENT packet must contain the same sequence number as the packet it is acknowledging.

Out of sequence DATA packets (any packet with a previously acknowledged sequence number or a packet whose sequence number is greater than the next expected sequence number) are discarded by the receiving MNCP. As part of the processing of out of sequence DATA packets, the receiving MNCP returns an ACKNOWLEDGEMENT packet containing the sequence number of the most recently acknowledged DATA packet.

The receiver processes the incoming DATA packets as follows. As part of the process of determining whether a properly composed DATA packet has arrived, the receiver checks to see if the correlation identifier in the packet corresponds to a transfer in progress; if the packet is incorrectly composed or the correlation identifier is unknown (not associated with a transfer in progress), the packet is discarded. If the DATA packet is valid, the receiver uses the packet contents

(correlation identifier, data offset, sequence number, more/final information, application data) and information relayed in the NOTIFICATION request (message length, compressed and uncompressed, compression method) to reassemble the application data. The receiver composes and returns an ACKNOWLDEGEMENT packet containing the correlation identifier and sequence number from the DATA packet being acknowledged. The ACKNOWLEDGEMENT packet is sent as a UDP packet to the source port which sent the DATA packet being acknowledged.

Reassembly of application data continues in this manner until a DATA packet containing a "final data" indicator is processed. The reassembled data are uncompressed (if compression was used). Application-specific information elements are forwarded to the application, and a final ACKNOWLEDGEMENT packet is sent as a UDP packet to the source port which sent the DATA packet being acknowledged.

2.3 Session Control Service

MNCP session control packets are exchanged between an MCD and a Mobility Server using MNCP reliable delivery packets. The purpose of session control is to provide user validation (authentication), application access control, user registration (deregistration) for application services, and application request/response correlation.

2.3.1 Subscriber Validation

User validation (authentication) is based on a subscriber identifier and subscriber password. A subscriber identifier is assigned to the

user of a mobile computing device, and is intended to be independent from any lower layer (e.g., IP) addressing or identification. In particular, access controls to applications and services may be based on subscriber identification, allowing the subscriber to access these applications irrespective of the IP or equivalent network address the user of an MCD is (temporarily) using for communication with a Mobility Server.

Applications and specific functions of applications that may be accessed by a user of a MCD (remotely invoked operations) are identified by a service identifier, which is globally unique and IANA-assigned, and a function identifier, which is service-specific. Application registration and deregistration are functions performed for all application services, and fall within session control, whereas other functions, such as a "check mailbox status" function, are specific to an application (mobile messaging), and are thus transparent to the MNCP.

2.3.2 Registration

Registration is a process whereby a client (MCD) notifies a Mobility Server of its intent to make use of an application service. An explicit form of registration is accomplished as follows. Session control information (subscriber identification and authentication information, application service and function identification) is supplied by the client application to the MCD's MNCP and encoded as information elements in a REGISTRATION request (see also section 2.1). A registration request is sent by a client (MCD) MNCP using the single packet reliable delivery mechanism.

The receiving MNCP (here, the Mobility Server) uses the subscriber identification and authentication information to validate the user and to determine whether the user has access privileges to the application service and function identified. The receiving (MS) MNCP returns a positive or negative ACKNOWLEDGMENT based on the success or failure of authentication and access control processing. If the authentication succeeds, the (MS) MNCP updates the status of the subscriber to "registered", records the IP address of the MCD from which the subscriber's registration request was initiated, and returns a positive ACKNOWLEDGEMENT. The (MS) MNCP starts a timer that bounds the amount of time the registered subscriber may be inactive before the subscriber is declared unavailable (see sections 4.7 and section 6.4).

[NOTE: Explicit registration may be used to enable "push" applications. Once a client application at an MCD is registered, an application at a

Mobility Server may send unsolicited messages to the MCD.]

A second, implicit form of registration occurs when an application at a Mobility Server receives requests for a service from a MCD that has not explicitly registered for that service. If the subscriber identification and authentication are valid, and access to this service is permitted for this subscriber, the MS will register the client (MCD) as previously described, and process the service request (see section 2.3.3 and section 3).

Once a subscriber has registered, a Mobility Server will forward all subsequent subscriber-bound messages to the MCD at the IP address recorded, until the subscriber explicitly deregisters the service, or registers the service from another MCD, or from a different IP address.

2.3.3 Application Data Transfer

Once registration is completed, applications at either the MCD or MS may begin sending requests. Session Control information (application service and function identification, subscriber identification and password, request/response message correlation information) accompany application-specific control information and data in each request. Each request (carried as a COMMAND packet or a NOTIFICATION packet initiating a multi-packet sequence) is authenticated. If authentication succeeds and all the contents are reliably delivered, a positive ACKNOWLEDGEMENT is returned to the MCD MNCP. Application-specific IE's as well as data are forwarded to the application when the entire request has been delivered.

2.3.4 Deregistration

Deregistration may be initiated by the MCD application. A request to deregister from an application service results in the transmission of a DEREGISTER function by session control to the Mobility Server's (MS) MNCP. Deregistration can also occur when an inactivity timer operating at the Mobility Server expires. When either of these events occurs, the (MS) MNCP deregisters the subscriber (i.e., changes the registration status to deregistered for the application service indicated in the request). The MNCP requesting deregistration (either

the MCD or MS) composes and sends a Deregistration request and awaits an indication from reliable transfer that

(a) the MCD MNCP has responded with an ACKNOWLDGEMENT indicating it wishes to continue communicating with the server. In this case, the (MS) MNCP updates the registration status of the client (to

registered).

(b) the (MCD or MS) MNCP has responded with an ACKNOWLEDGEMENT indicating it agrees to discontinue communication. In this case, the requesting MNCP remains in an unregistered (i.e., inactive) status.

(c) retry timer expiration causes the reliable delivery MNCP to indicate to session control that communication attempts have been abandoned. In this case, the requesting MNCP remains in an unregistered (i.e., inactive) status.

2.3.5 Correlation Identifiers

The correlation identifier is used by Session Control to associate packets (or packet sequences) of a given exchange, and is relevant for both directions of information flow (i.e., the acknowledgment for a NOTIFICATION, COMMAND, and DATA packet must have the same correlation identifier) for the duration of that exchange. The correlation identifier has local significance to the mobile computing device or Mobility Server.

Applications may use a Correlation Identifier value to link together or "cross-correlate" packet sequences related to the same application-specific message flow. Consider an e-mail client application request from a MCD to retrieve mailbox messages. The request could be satisfied using a single COMMAND-ACK sequence. The corresponding responses could be conveyed as a packet sequence involving the use of the NOTIFICATION service initiated by a messaging application operating on a Mobility Server. The Cross-correlation Identifier value used by the Mobility Server when delivering the contents of the mailbox to the email client must be the same as the Correlation Identifier of the original request to retrieve mailbox messages (see section 4.5.5).

3. MNCP Reliable Delivery Packets

This section defines the packets which support MNCP reliable delivery services.

3.1 Packet Types

MNCP reliable delivery packets are exchanged between an MCD and a Mobility Server. There are four types of MNCP reliable delivery packets, differentiated by the Packet Type field in the Packet Header.

Command Packet (PT_CMD)
  This packet is used when the entire length of the application data can be carried in a single packet.

Notification Packet (PT_NTFN)
This packet is used when the entire length of the user data can not fit into a single packet. It is followed by one or more PT_DATA packets.

Data Packet (PT_DATA)
This packet is used in conjunction with the Notification Packet to carry the actual application data.

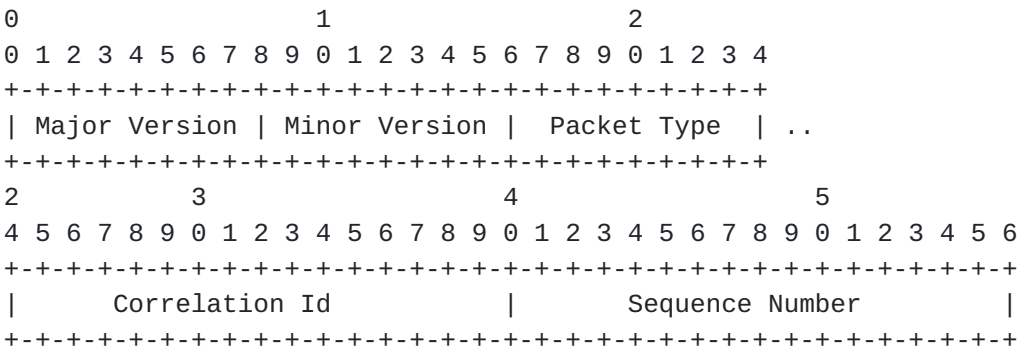
Acknowledge Packet (PT_ACK)
This packet is used to confirm receipt of a PT_CMD, PT_NTFN, or PT_DATA packet by the peer MNCP.

MNCP reliable delivery packets PT_CMD, PT_NTFN, and PT_DATA can originate from either the MCD or the Mobility Server. Acknowledgments (PT_ACKs) are returned by the recipient of the other packets.

Exactly one MNCP reliable delivery packet is encapsulated in each UDP Information field as an octet sequence, encoded in network-byte order.

3.2 Packet Headers

The following common header fields appear in every MNCP reliable delivery packet. A summary of the packet header format is shown below. The bits are transmitted in network-byte order, from left to right.



Major and Minor Protocol Version
Two one octet Protocol Version fields identify the Major and Minor Version level of the MNCP packet. The values (1,1) MUST be used to indicate the MNCP protocol version specified by this memo. When a packet is received with an unknown Protocol Version value, the packet SHOULD be silently discarded.

Packet Type
The Packet Type field is one octet, and identifies the type of MNCP reliable delivery packet as enumerated below.

1 PT_CMD Command

2	PT_NTFN	Notification
3	PT_DATA	Data
4	PT_ACK	Acknowledge

When a packet is received with an unknown Packet Type value, the packet SHOULD be silently discarded.

Correlation Id

The Correlation Id field is two octets, and is used to link together all the packets in a particular packet sequence. When initiating a new packet sequence, applications at a Mobility Server MUST select values in the range h0001 - h7FFF (i.e., the most significant bit must be zero, 0). When initiating a new packet sequence, applications at a MCD MUST select values in the range h8000-hFFFF (i.e., the most significant bit must be one, 1). The value zero is reserved and MUST NOT be used.

When a PT_DATA or PT_ACK packet is received with an unknown Correlation Identifier field, or any other type of packet is received with a missing Correlation Identifier field, the packet SHOULD be silently discarded.

The MNCP that initiates a packet sequence (i.e., sends a PT_CMD or PT_NTFN packet) MUST ensure that the Correlation Identifier value uniquely identifies the packet sequence locally (i.e., no other packet sequence is in progress involving this MCD and the same Correlation Identifier value). All other packets in this packet sequence (including PT_ACKs) MUST carry this same Correlation Identifier value.

Applications may use the same Correlation Identifier value to link together packet sequences related to the same application-specific message flow. For example, an e-mail client request might be conveyed by a PT-CMD packet sequence, with mail server responses conveyed as PT-NTFN packets sequences, all of which carry Cross Correlation Identifiers equal to the Correlation Identifier of the original request.

Sequence Number

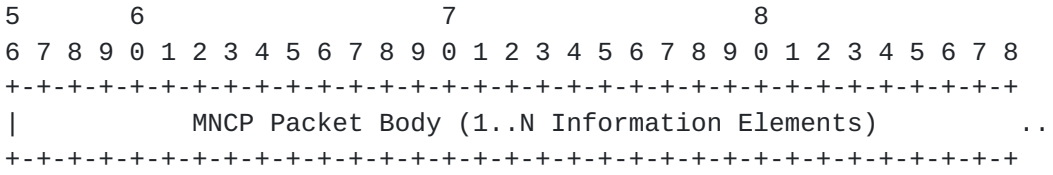
The Sequence Number field is two octets, and is used to maintain sequencing of packets. When a packet is received with a missing or unknown Sequence Number field, the packet SHOULD be silently discarded.

The sequence number MUST have the value zero (0) for the first packet

of a packet sequence. The sequence number contained in each subsequent PT_DATA packet within the same packet sequence MUST be incremented sequentially by one (1) until the maximum field value (65,536) has been reached, and then recycled back to zero(0).

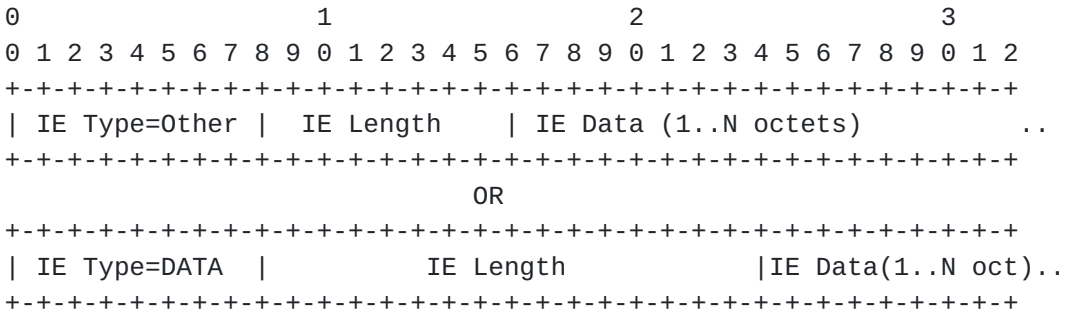
3.3 Packet Body

The body an MNCP packet is used to carry MNCP reliable delivery control information, session control information, and application-specific data. The MNCP packet body is transmitted immediately after the MNCP packet header, beginning at bit 56 of the entire MNCP reliable delivery packet.



The MNCP reliable delivery packet body consists of one or more Information Elements (IEs). Each Information Element consists of an IE Type field, IE Length field, and a variable-length data field (content determined by the IE Type, length identified by the IE Length). There are two Information Element formats: an extended length format used only for IE_DATA_MORE and IE_DATA_FINAL elements, and an abbreviated format used for all other currently-defined IE Types.

A summary of the Information Element field format is shown below. The fields are transmitted in network-byte order, from left to right, beginning with the first available bit after the MNCP packet header or preceding Information Element.



IE Type
The IE Type field is one octet, and identifies the type of Information Element as enumerated below.

5	IE_DATA_FINAL	Data (Final)
6	IE_DATA_MORE	Data (More)
8	IE_MSG_LENGTH	Message Length
10	IE_ACK_CODE	Acknowledge Code
16	IE_DATA_COMPRESSION	Data Compression
18	IE_DATA_OFFSET	Data Offset
20	IE_PKT_SIZE	Packet Size

Additional IE Type values are defined by MNCP Session Control (see Section 4.5) and may also be defined for use by specific applications (to be assigned and recorded by IANA[2]). When a packet is received with an unknown IE Type value, the Information Element SHOULD be forwarded to the application without further interpretation by the MNCP.

IE Length
If the IE Type equals IE_DATA_MORE or IE_DATA_FINAL, the IE Length field is two octets; otherwise, the IE Length field is one octet. The IE Length field indicates the length of the information element data field, in octets.

Octets beyond the range of the IE Length field are treated as a separate Information Element. When a packet is received with an invalid Length field, the packet SHOULD be silently discarded.

IE Data
The format of the IE Data field varies according to IE Type. The format associated with each IE Type is defined in Sections 3.5 and 4.5.

When encoding MNCP packets, the following general rules apply, in order of priority.

- Required IEs MUST appear before optional IEs, and
- Fixed-length IEs MUST appear before variable-length IEs.

3.4 Packet Length

The length of each MNCP reliable delivery packet is the sum of the following:

MNCP Header Length	7 octets
MNCP Body Length	sum of Information Element lengths

When IE Type is IE_DATA_MORE or IE_DATA_FINAL, the length of the Information Element is three(3) octets plus the value specified by the IE Length field. Otherwise, the length of the Information Element is two (2) octets plus the value specified by the IE Length field. Since every MNCP reliable delivery packet contains at least one Information Element, the minimum length of a packet is 10 octets.

The maximum length of an MNCP packet, MAX_PACKET_SIZE, is limited to 2048 octets. The default packet size, DEFAULT_PACKET_SIZE, is 470 octets, chosen because it is the most efficient size that can be transported by UDP over wireless Mobitex networks. In order to increase network efficiency, the sending MNCP may propose a packet length greater than DEFAULT_PACKET_SIZE, but less than or equal to MAX_PACKET_SIZE. The receiving MNCP may accept the proposed value or request a smaller packet length. The selection of an appropriate packet size is affected by factors such as the Maximum Transmission Unit (MTU) and Maximum Segment Size (MSS) of the underlying network.

When an application message is longer than the negotiated packet size (less header and protocol control information overhead), it MUST be segmented into more than one MNCP reliable delivery packet. In this case, the length of the complete application message is indicated by the IE_MSG_LENGTH Information Element included in the PT_NTFN packet (see Section 3.5.2).

3.5 Information Elements

Each Information Element includes IE Type and IE Length fields, formatted as described in Section 3.3. Information Elements related to reliable transfer are defined below; additional elements related to session control are defined in Section 4.5.

When a packet is received with a syntactically invalid Information Element, the packet MUST be acknowledged with the Ack Code ACK_ERR_INFO. When a packet is received without a required Information Element, the packet MUST be acknowledged with the Ack Code ACK_ERR_PROT.

3.5.1 Data (Final and More)

0 1 2 3

```
0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|T=IE_DATA_FINAL| Length=1..N                | Data (1..N) ..
|or IE_DATA_MORE| where N = (MAX_PACKET_SIZE - header length)
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Data
  The Data field is variable length, ranging from one to (MAX_PACKET_SIZE - header) octets, and carries application-dependent content.

  The IE Type MUST equal IE_DATA_FINAL if the data field contains the only or final segment of an application message. Otherwise, the IE Type MUST equal IE_DATA_MORE, indicating that the remainder of the application message will be sent in subsequent PT_DATA packets.

  In a PT_DATA packet, the content of the data field may be compressed (see Section 3.5.4).

3.5.2 Message Length

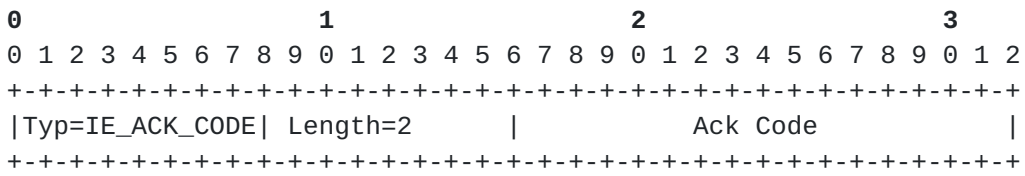
```
0                   1
0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|T=IE_MSG_LENGTH| Length=8      |..
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
1       2                   3                   4
6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                   Original Message Length                     |..
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
4   5                   6                   7
8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                  Compressed Message Length                    |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Original Message Length
  The Original Message Length field is four octets, and identifies the length, in octets, of the original application message to be transferred during an MNCP reliable delivery packet sequence. Providing this value allows the receiving MNCP to allocate adequate buffer space in which to build the decompressed message.

Compressed Message Length
  The Compressed Message Length field is four octets, and identifies the length, in octets, of the same application message after compression.

This indicates the number of data octets to be carried by the sequence of PT_DATA packets which follow this PT_NTFN packet, and may equal Original Message Length if no compression algorithm has been negotiated. Providing this value allows the receiving MNCP to allocate adequate buffer space in which to reassemble the compressed message.

3.5.3 Acknowledge Code

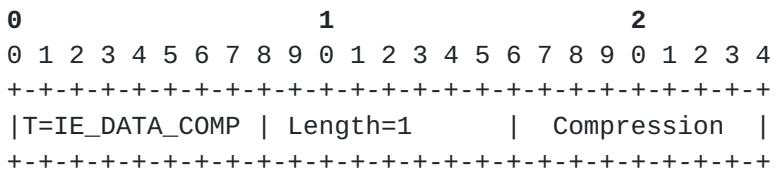


Ack Code

The Ack Code field is two octets, and indicates success or failure of MNCP packet processing. Values associated with reliable transfer processing are enumerated below; additional session control-related values are enumerated in Section 4.5.6.

ACK_OK	0	Success, no error
ACK_ERR_MCD	1	Unrecognized MCD
ACK_ERR_FILE_IO	9	Storage or File I/O error
ACK_ERR_INFO	11	Invalid parameters/command syntax
ACK_OOS_COMPRESS	12	Compression method not supported
ACK_ERR_PROT	13	Protocol Error
ACK_ERR_SYS	65535	Unrecoverable System Error

3.5.4 Data Compression



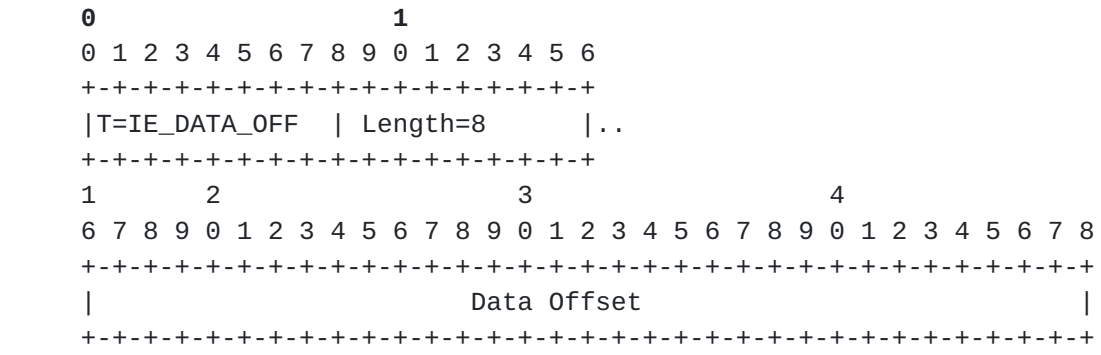
Compression

The Compression field is one octet, and identifies the compression method to be used on the data contained within PT_DATA packet IE_DATA_MORE and IE_DATA_FINAL information elements, as enumerated below.

COMPRESS_OFF	0	Data MUST NOT be compressed
COMPRESS_DEFAULT	1	Data MUST be compressed using default method, LZS, as defined by RFC 1974[3]

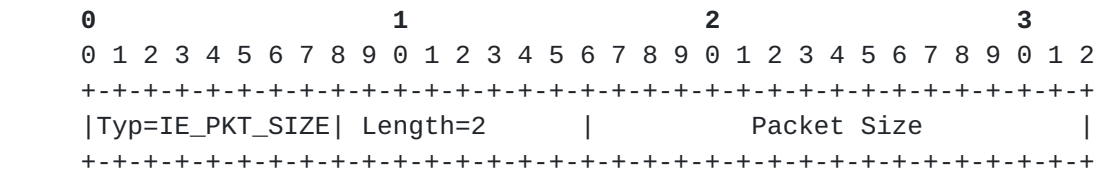
Additional values for this field may be assigned and recorded by IANA[2]. Section 5.8 describes how this field is used for negotiation in PT_NTFN and PT_ACK packets.

3.5.5 Data Offset



Data Offset
The Data Offset field is four octets. When uncompressed data are sent, Data Offset identifies the offset, in octets, of the first bit of the IE_DATA_MORE or IE_DATA_FINAL Data field from the beginning of the original uncompressed message. When compression is used, Data Offset identifies the offset, in octets, of the first bit of the IE_DATA_MORE or IE_DATA_FINAL Data field from the beginning of the compressed message.

3.5.6 Packet Size



Packet Size
The Packet Size field is two octets, and identifies the maximum size (in octets) for PT_DATA packets in this MNCP packet sequence. Section 5.7 describes how this field is used for negotiation in PT_NTFN and PT_ACK packets and how it affects segmentation of application data in PT_DATA packets.

The default value for this field is DEFAULT_PACKET_SIZE (470 octets). The maximum value for this field is MAX_PACKET_SIZE (2048 octets). If this field is absent from a PT_NTFN packet, the default value MUST be assumed.

3.6 PT_CMD

When an application wishes to send a message that is shorter than (DEFAULT_PACKET_SIZE - header) octets, uncompressed, the MNCP MUST embed it in the IE_DATA_FINAL field of a PT_CMD packet.

Upon reception of a correctly-formed PT_CMD packet, a PT_ACK MUST be transmitted, containing a positive or negative Ack Code as described in Section 3.7.

The following IEs are optional in a PT_CMD packet.

IE_DATA_FINAL	As defined in Section 3.5.1 MUST be present if application data has been supplied by the user.

Additional session control or application-specific IEs may also be included in the PT-CMD packet.

3.7 PT_ACK

Upon reception of a correctly-formed PT_CMD, PT_NTFN, or PT_DATA packet, a PT_ACK MUST be transmitted to confirm delivery. The PT_ACK includes the same Correlation ID and Sequence Number as the packet to be confirmed, and an Ack Code to indicate the success or failure of packet processing within the MNCP.

The following IE is required in a PT_ACK packet.

IE_ACK_CODE	As defined in Section 3.5.3

Additional session control IEs and application-specific IEs may also be included in the PT_ACK packet.

3.8 PT_NTFN

When an application wishes to send a message that is longer than (DEFAULT_PACKET_SIZE - header) octets, uncompressed, the MNCP MUST generate a PT_NTFN packet to initiate a sequence of PT_DATA packets.

Upon reception of a correctly-formed PT_NTFN packet, a PT_ACK MUST be transmitted, containing a positive or negative Ack Code as described in Section 3.7.

The following IEs are required in a PT_NTFN packet.

IE_MSG_LENGTH	As defined in Section 3.5.2

The following IEs are optional in a PT_NTFN packet.

IE_DATA_COMPRESSION	As defined in Section 3.5.4 MUST be present if compression desired otherwise default value assumed
IE_PKT_SIZE	As defined in Section 3.5.6 MUST be present if long packets desired otherwise default value assumed

Additional application-specific IEs may also be included in the PT-NTFN packet.

3.9 PT_DATA

When an application wishes to send a message that is longer than (DEFAULT_PACKET_SIZE - header) octets, uncompressed, the MNCP generates a sequence of PT_DATA packets to carry message segments. The final PT_DATA packet carries the information element IE_DATA_FINAL; all others carry the information element IE_DATA_MORE. These packets carry the same Correlation ID and sequentially-assigned Sequence Numbers.

Upon reception of each correctly-formed PT_DATA packet, a PT_ACK MUST be transmitted, containing a positive or negative Ack Code as described in Section 3.7.

The following IEs are required in a PT_DATA packet.

IE_DATA_OFFSET	As defined in Section 3.5.5
IE_DATA_MORE or IE_DATA_FINAL	As defined in Section 3.5.1

4. MNCP Session Control Packets

This section defines the packets which support MNCP session control services.

4.1 Packet Types

MNCP session control packets are exchanged between an MCD and a Mobility Server using MNCP reliable delivery packets. There are three types of MNCP session control packets, differentiated by the Function ID field of the IE_APP_ID information element in the MNCP session control header.

Registration Packet (PT_CMD, Function ID = FUN_REG_REQ)
This packet is used to register a Subscriber to use the specified Service.

Deregistration Packet (PT_CMD, Function ID = FUN_DEREG_REQ)
This packet is used to deregister a Subscriber so that it can no longer use the specified Service.

Application Request Packet (PT_CMD or PT_NTFN, Function ID = other)
This packet is used to request an application-specific service, specified by Service ID and Function ID.

Registration packets originate from the MCD, to be processed and acknowledged by the Mobility Server. Deregistration and Application Request packets can be initiated by either the MCD or Mobility Server.

Exactly one MNCP session control packet is conveyed by each PT_CMD or PT_NTFN packet, utilizing those fields previously defined for MNCP reliable delivery, and the additional Session Control fields defined in this section.

4.2 Session Control Headers

Information Element types defined specifically for use as MNCP Session Control header fields are enumerated below and defined in Section 4.5.

1	IE_SUB_ID	Subscriber ID
3	IE_APP_ID	Application ID
9	IE_SUB_PWD	Subscriber Password

These Information Elements are mandatory in every MNCP session control packet, regardless of Service ID or Function ID, and may appear anywhere within the list of Information Elements in an MNCP packet.

4.3 Session Control Body

Information Element types defined specifically for use as MNCP Session Control body fields are enumerated below and defined in Section 4.5.

11	IE_REG_STATUS	Registration Status
24	IE_CROSS_ID	Message Cross-correlation ID

These Information Elements may be included in certain MNCP session control packets, as determined by Function ID, and may appear anywhere within the list of Information Elements in an MNCP packet.

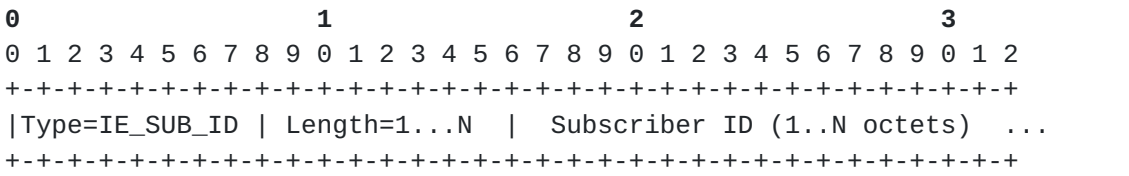
4.4 Packet Length

The length of the MNCP session control packet can be computed as the sum of the lengths of each session control Information Element.

4.5 Information Elements

Each Information Element includes IE Type and IE Length fields, formatted as described in Section 3.3. Information Elements related to reliable transfer are defined in Section 3.5; additional elements related to session control are defined below.

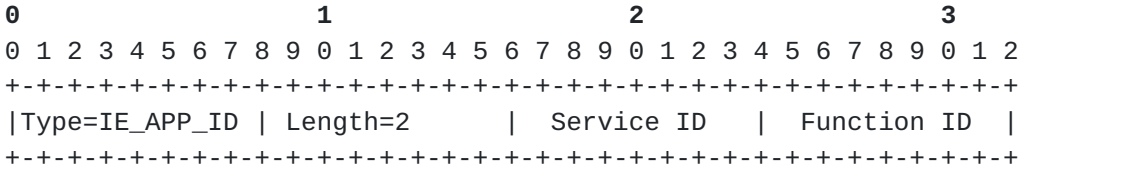
4.5.1 Subscriber ID



Subscriber ID

The Subscriber ID field is variable length, and identifies the user of the MCD. This value is used by MNCP Session Control to provide subscriber validation/authentication (see Section 7).

4.5.2 Application ID



Service ID

The Service ID field is one octet, and identifies the Information Service (specific application) involved in this MNCP packet sequence. This field identifies both the application which originated the message and the destination application that is to receive the message.

Service ID values are assigned and recorded by IANA[2] for use by specific applications. Service ID is used by MNCP Session Control to provide service registration, deregistration, and filtering features.

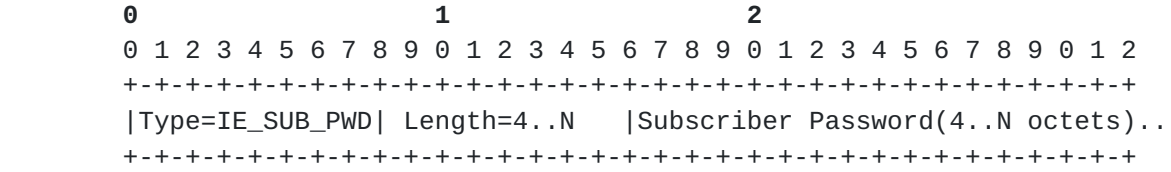
Function ID

The Function ID field is one octet, and identifies the function within the specified Service requested by this MNCP packet, as enumerated below.

0	FUN_DEREG_REQ	Deregistration Request
1	FUN_REG_REQ	Registration Request
2..255	TBD	Application-Dependent

Function ID values zero (0) and one (1) are reserved for use by MNCP Session Control. Function ID values 2 through 255 are application-dependent, and are transparent to the MNCP.

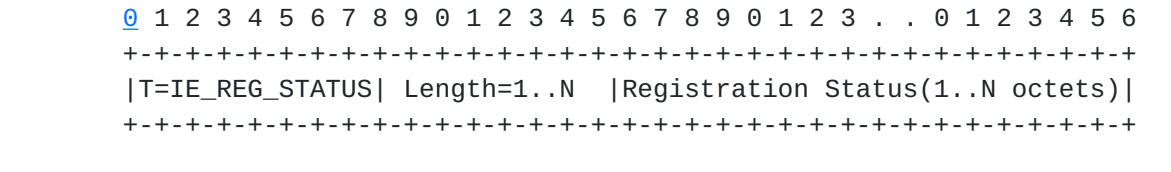
4.5.3 Subscriber Password



Subscriber Password

The Subscriber Password field is variable length, ranging from four to N octets, and carries a value used by MNCP Session Control for user validation/authentication (see Section 7).

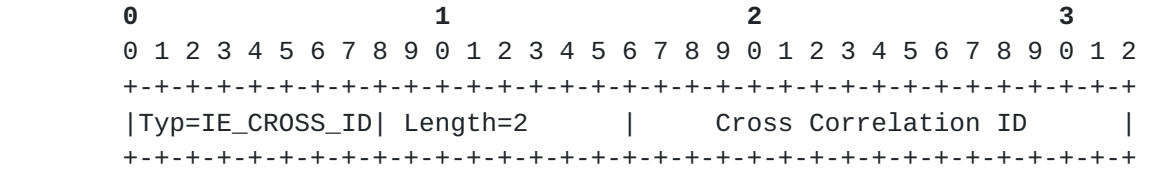
4.5.4 Registration Status



Registration Status

The Registration Status field is a variable length field, where each octet contains the ID of a Service for which this Subscriber is currently registered.

4.5.5 Message Cross-Correlation ID



Message Cross-correlation ID

The Message Cross-correlation ID is two octets, and carries a 16-bit unsigned integer number identifying the correlation ID of a previous message flow to which the current message flow is associated. Using a cross-correlation identifier, a receiving application correlates a response message with a previous request message.

4.5.6 Acknowledge Code

MNCP Session Control defines additional values for the Ack Code field specified in Section 3.5.3.

ACK_ERR_SID	2	Unrecognized Subscriber ID
ACK_ERR_PWD	3	Incorrect Password
ACK_OOS_SID	5	Subscriber ID is suspended
ACK_OOS_SVC	10	Application service unavailable

4.6 FUN_REG_REQ

When an application wishes to explicitly register for a specific Service ID (see section 2.3.2), it MUST first generate a Registration Request by issuing a PT_CMD packet with the desired Service ID and Function ID set to FUN_REG_REQ.

Upon reception of a Registration Request, MNCP session control performs authentication based upon Service ID, Subscriber ID, and Password, marking the subscriber as "registered" and recording the IP address of the MCD from which the request was received if the subscriber is determined to be authentic and authorized to use the service. A PT_ACK MUST be transmitted in response, containing a positive or negative Ack Code as described in Section 4.5.6. In addition, as an implementation option, the PT_ACK MAY contain the IE_REG_STATUS Information Element described in Section 4.5.4.

4.7 FUN_DEREG_REQ

When an application wishes to stop using a specific Service ID, it may generate a Deregistration Request by issuing a PT_CMD packet with the desired Service ID and Function ID set to FUN_DEREG_REQ.

The MCD MNCP MAY deregister from a service by generating a Deregistration Request, either implicitly on behalf of the application, explicitly upon application request, or both. Upon receiving an MCD-initiated Deregistration Request, the MS MNCP session control performs authentication based upon Service ID, Subscriber ID, and Password, marking the subscriber as "deregistered" if authorized to use the service. A PT_ACK MUST be transmitted in response, containing an Ack Code as described in Section 4.5.6.

An MS MNCP that has not detected activity on the part of a registered subscriber for a given period of time (INACTIVITY_TIMER) MAY attempt to deregister the subscriber implicitly by generating a Deregistration Request. Upon receiving an MS-initiated Deregistration Request, the MCD MNCP determines whether the application identified by the service ID is still running. If so, the MCD MNCP MUST return a PT_ACK (ACK_OK)

and the MS MNCP MUST re-register the subscriber. Otherwise, the MCD MNCP returns a PT_ACK (ACK_OOS_SVC) and the MS interprets this as a confirmation that the application service is no longer running on the subscriber's MCD.

In addition, as an implementation option, these PT_ACKs MAY contain the IE_REG_STATUS Information Element described in Section 4.5.4. As a security precaution, this field SHOULD NOT be included when acknowledging a Deregistration Request that has not passed authentication.

4.8 FUN_<other>

When an application wishes to send a request with a specific Service ID, it MUST generate an Application Request by issuing a PT_CMD or PT_NTFN packet with the desired Service ID and Function ID.

Upon reception of an Application Request, and if the subscriber is currently registered to use the service, MNCP session control performs authentication based upon Service ID, Subscriber ID, and Password, and forwards the request to the destination application.

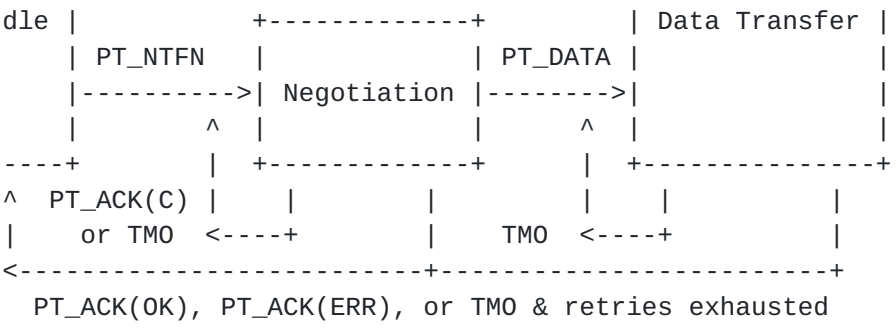
If the subscriber is not currently registered to use the service, MNCP session control performs authentication based upon Service ID, Subscriber ID, and Password. If the subscriber is determined to be authentic and permitted to access the service, MNCP session control marks the subscriber as "registered" and forwards the request to the destination application (implicit registration).

For either case, a PT_ACK MUST be transmitted in response, containing a positive or negative Ack Code as described in Section 4.5.6. A positive Ack Code indicates that the request can be delivered to the destination application; a negative Ack Code indicates why the request cannot be delivered.

5. MNCP Reliable Delivery Processing

5.1 Phase Diagram

MNCP reliable delivery goes through three distinct phases which are specified in the following simplified diagram.



Idle Phase

The MNCP necessarily begins and ends with this phase. When an application or session control request causes a PT_NTFN packet to be sent or received, the MNCP proceeds to the Negotiation phase. When an application or session control request causes a PT_CMD packet to be sent or received, the MNCP proceeds to the Data Transfer phase.

The MNCP returns to the Idle phase automatically if:

- the ack timeout expires while waiting for any packet (i.e., packet lost or silent discard of badly-formed packet) and all retries have been exhausted;

- a PT_ACK packet with IE_ACK_CODE equal to ACK_OK is sent or received confirming the final packet in a packet sequence (i.e., confirming a PT_DATA packet that contained an IE_DATA_FINAL element); or

- a PT_ACK packet with IE_ACK_CODE not equal to ACK_OK is sent, or a PT_ACK packet with IE_ACK_CODE not equal to ACK_OK or ACK_OSS_COMPRESS is received.

Negotiation Phase

The MNCP enters this phase when sending or receiving a PT_NTFN packet.

The receiving MNCP processes the incoming packet and returns a positive or negative PT_ACK as follows.

- If the PT_ACK is negative, the receiver returns to the Idle phase.

- If the PT_ACK is positive, the receiver enters the Data Transfer phase.

The sending MNCP awaits and processes the PT_ACK.

- If the PT_ACK is positive, the sender enters the Data Transfer phase.

- If the PT_ACK indicates ACK_OSS_COMPRESS, the sender SHOULD retry sending the PT_NTFN with a different compression method and remain in the Negotiation phase.

- Otherwise, the sender returns to the Idle phase.

Any locally-initiated application request received during this phase MUST NOT be processed by this MNCP instance until the Idle Phase is re-entered. Such requests MAY be rejected or buffered locally by the implementation.

Data Transfer Phase

The MNCP enters this phase when sending or receiving a PT_CMD or PT_DATA packet.

The receiving MNCP processes the incoming packet and returns a positive or negative PT_ACK as follows.

- If the PT_ACK is negative or confirms receipt of a packet containing an IE_DATA_FINAL element, the receiver returns to the Idle phase.

- Otherwise (confirming PT_DATA packet with IE_DATA_MORE), the receiver remains in the Data Transfer phase.

The sending MNCP awaits and processes the PT_ACK.

- If PT_ACK is negative or the outgoing packet contained an IE_DATA_FINAL element, the sender returns to the Idle phase.

- Otherwise (PT_ACK confirmed packet containing IE_DATA_MORE), the sender remains in the Data Transfer phase and sends the next PT_DATA packet.

Any locally-initiated application request received during this phase MUST NOT be processed by this MNCP instance until the Idle Phase is re-entered. Such requests MAY be rejected or buffered locally by the

implementation.

5.2 State Diagram

The MNCP reliable delivery finite-state automaton is defined by events, actions and state transitions. Events include reception of application requests, expiration of the Ack timer, and reception of packets from a peer. Actions include the starting of the timers and transmission of packets to the peer.

Events	Actions
REQ = Receive Application Request	scm = send PT_CMD
RCM = Receive PT_CMD	snt = send PT_NTFN
RNT = Receive PT_NTFN	fsc = fwd to Session Control
RSP = Receive Session Response	sak = send PT_ACK
RAK = Receive PT_ACK	sdt = send PT_DATA
RDT = Receive PT_DATA	sto = start timers
TMO = Ack Timeout	ind = send Appl Indication
	cnf = send Appl Confirm

The complete state transition table follows. States are indicated horizontally, and events are read vertically. State transitions and actions are represented in the form action/new-state. Multiple actions are separated by commas, and may continue on succeeding lines as space requires; multiple actions may be implemented in any convenient order. The state may be followed by a letter, which indicates an explanatory footnote. The dash ('-') indicates an illegal transition.

	State					
	0	1	2	3	4	5
Events	Listen	Cmd-Sent	Ntfn-Sent	Data-Sent	Await-Rsp	Await-Data

REQ	scm/1 or snt/2	-	-	-	-	-
RCM	sak/0 or fsc/4	-	-	-	-	-
RNT	sak/0 or fsc/4	-	-	-	-	-
RSP	-	-	-	-	sak/0 or sak/5 or ind,sak/0	-
RAK	-	cnf/0	cnf/0 or snt/2 sdt/3	cnf/0 or sdt/3	-	-
RDT	-	-	-	-	-	sak/5 or ind,sak/0
TMO	-	cnf/0 or scm/1	cnf/0 or snt/2	cnf/0 or sdt/3	-	0

Timers are started (sto action) when sending any packet and stopped when receiving any packet. The sending MNCP runs an ACK_WAIT_TIMER; the receiving MNCP runs a DATA_WAIT_TIMER. See section 5.6 for additional detail.

5.3 States

Following is a more detailed description of each automaton state.

Listen
: The MNCP automaton begins and ends in this state, awaiting either a locally-initiated application or session control request, or receipt of a PT_CMD or PT_NTFN packet.

Command-Sent
: The MNCP automaton transitions to this state when sending a PT_CMD packet. Events expected to occur while in this state are expiration of the ACK_WAIT_TIMER or receipt of a PT_ACK packet.

Notification-Sent
: The MNCP automaton transitions to this state when sending a PT_NTFN packet. Events expected to occur while in this state are expiration of the ACK_WAIT_TIMER or receipt of a PT_ACK packet.

Await-Data
: The MNCP automaton transitions to this state when sending a positive PT_ACK packet in response to an incoming PT_NTFN packet. Events expected to occur while in this state are expiration of the DATA_WAIT_TIMER or receipt of a PT_ACK packet.

Await-Session-Response
The MNCP automaton transitions to this state when forwarding an incoming PT_CMD or PT_NTFN packet to MNCP session control. The only event expected to occur while in this state is a response from session control.

Data-Sent
The MNCP automaton transitions to this state when sending a PT_DATA packet. Events expected to occur while in this state are expiration of the ACK_WAIT_TIMER or receipt of a PT_ACK packet.

5.4 Events

Transitions and actions in the automaton are caused by events.

Receive Application Request (REQ)
This event occurs when a locally-initiated application or session control request is submitted to the MNCP for processing. If the data contained in the request is shorter than (DEFAULT_PACKET_SIZE - header) octets, uncompressed, the MNCP automaton sends a PT_CMD (scm action). Otherwise, it sends a PT_NTFN (snt action).

Receive PT_CMD (RCM)
This event occurs when a remotely-initiated PT_CMD packet is received by the MNCP as an incoming UDP datagram. If the packet is badly formed, contains an invalid version, a system error occurs, or resources are unavailable to further process the request, the MNCP automaton sends a PT_ACK (sak action) with a negative Ack Code. Otherwise, it forwards the PT_CMD packet to the MNCP session control automaton (fsc action).

Receive PT_NTFN (RNT)
This event occurs when a remotely-initiated PT_NTFN packet is received by the MNCP as an incoming UDP datagram. If the packet is badly formed, contains an invalid version, a system error occurs, resources are unavailable to further process the request, or the proposed compression method is not supported, the MNCP automaton sends a PT_ACK (sak action) with a negative Ack Code, and an alternative compression method (if applicable). Otherwise, it forwards the PT_NTFN packet to the MNCP session control automaton (fsc action).

Receive Session Response (RSP)
This event occurs when a response is returned from the local MNCP session control automaton, indicating the success or failure of authentication, registration, or deregistration. If the response is negative, the MNCP automaton sends a PT_ACK (sak action) with the

negative Ack Code supplied by session control. If the response is positive, the MNCP automaton sends a PT_ACK (sak action) ACK_OK. If the incoming request was a PT_CMD packet, the MNCP automaton supplies any application-specific information elements (including data) to the destination application (ind action).

Receive PT_ACK (RAK)
This event occurs when a remotely-initiated PT_ACK packet is received by the MNCP as an incoming UDP datagram. If the Ack Code is

ACK_OOS_COMPRESS, the MNCP automaton SHOULD resend the PT_NTFN with another compression method (snt action). If the Ack Code is any other negative value, the MNCP automaton notifies the requesting application of the failure (cnf action). If the Ack Code is ACK_OK and confirms a PT_NTFN or PT_DATA(more) packet, the MNCP automaton sends a PT_DATA packet (sdt action). Otherwise (the Ack Code is ACK_OK and confirms the final packet in a sequence), the MNCP automaton notifies the requesting application that its request was delivered successfully (cnf action).

Receive PT_DATA (RDT)
This event occurs when a remotely-initiated PT_DATA packet is received by the MNCP as an incoming UDP datagram. The MNCP automaton uses the packet contents to reassemble and decompress application data (see Sections 5.6 through 5.8) and send a PT_ACK (sak action). If incoming packet contained an IE_DATA_FINAL element, the MNCP automaton supplies all application-specific information elements received during the packet sequence, including the reassembled/decompressed application data, to the destination application (ind action).

Ack Timeout (TMO)
This event occurs when the ACK_WAIT_TIMER or DATA_WAIT_TIMER started by this MNCP automaton expires. The automaton applies its retransmission algorithm (see Section 5.6) to determine the appropriate action: resending a PT_CMD packet (scm action), PT_NTFN packet (snt action), PT_DATA packet (sdt action), or PT_ACK packet (sak action), or abandoning the packet sequence. When abandoning the packet sequence, if this MNCP was the sequence initiator, it also notifies the requesting application of the failure (cnf action).

5.5 Actions

Actions in the automaton are caused by events and typically indicate the transmission of packets and/or the starting or stopping of the timers.

send PT_CMD (scm)
The MNCP automaton sends a PT_CMD packet as a result of a locally-initiated application or session control request, or as a retry (ACK_WAIT_TIMER expired with retry remaining).

The MNCP builds a PT_CMD packet as follows.
- Protocol Version is set to the value for this implementation.
- Packet Type is set to PT_CMD.
- Correlation Identifier is assigned by the MNCP (new request) or set to the previously-assigned value (retry).
- Sequence Number is set to zero(0).
- Session Control fields supplied in the request (Service ID, Function ID, Subscriber ID, Subscriber Password) are appended.
- Any application-specific IEs supplied in the request are appended.
- Any application-specific data supplied in the request is encapsulated in a Data(Final) information element.

The PT_CMD packet is sent as a UDP packet to the "well-known" MNCP port, the PKT_RETRY count is incremented, and the ACK_WAIT_TIMER is

started (see Section 5.6). The MNCP automaton then transitions to the Command-Sent state.

send PT_NTFN (snt)
The MNCP automaton sends a PT_NTFN packet as a result of a locally-initiated application or session control request, or as a retry (ACK_WAIT_TIMER expired with retry remaining or PT_ACK ACK_OOS_COMPRESS received and alternative method available).

The MNCP builds a PT_NTFN packet as follows.
- Protocol Version is set to the value for this implementation.
- Packet Type is set to PT_NTFN.
- Correlation Identifier is assigned by the MNCP (new request) or set to the previously-assigned value (retry).
- Sequence Number is set to zero(0).
- Data Compression Method is chosen by the MNCP, influenced by values returned in earlier PT_ACKs (if any); see Section 5.8. If the default value (COMPRESS_OFF) is selected, this field SHOULD be omitted.
- Uncompressed Message Length is set to the number of octets of application-specific data supplied in the request.
- Compressed Message Length is set to the number of octets yielded when compressing this data using the proposed Compression Method.
- As an implementation option, Packet Size may be set to a value larger than DEFAULT_PACKET_SIZE; see Section 5.7. If the default

value is desired, this field SHOULD be omitted.
- Session Control fields supplied in the request (Service ID, Function ID, Subscriber ID, Subscriber Password) are appended.
- Any application-specific IEs supplied in the request are appended.

The PT_NTFN packet is sent as a UDP packet to the "well-known" MNCP port, the PKT_RETRY count is incremented, and the ACK_WAIT_TIMER is started (see Section 5.6). The MNCP automaton then transitions to the Notification-Sent state.

fwd to Session Control (fsc)
The MNCP automaton forwards incoming PT_CMD and PT_NTFN packets to MNCP session control for registration, deregistration, and authentication. The MNCP automaton then transitions to the Await-Session-Response state.

send PT_ACK (sak)
The MNCP automaton sends a PT_ACK packet in response to an incoming packet.

The MNCP builds a PT_ACK packet as follows.
- Protocol Version is set to the value for this implementation.
- Packet Type is set to PT_ACK.
- Correlation Identifier and Sequence Number are both set to the corresponding values in the packet being acknowledged.
- Ack Code is chosen by the MNCP to reflect the result of processing.

When acknowledging a PT_NTFN only, the following fields MAY also be included in the PT_ACK packet.
- If the Ack Code is ACK_OOS_COMPRESS, an alternative Data Compression Method MUST be chosen by the MNCP; see Section 5.8. Otherwise, this field MUST be omitted.
- As an implementation option, PACKET_SIZE MAY be set to a value larger than DEFAULT_PACKET_SIZE and less than the proposed size, see Section 5.7. If the default value is desired, this field SHOULD be omitted.

The PT_ACK packet is sent as a UDP packet to the source port which sent the packet being acknowledged. When positively acknowledging a PT_NTFN or PT_DATA(more) packet, the MNCP automaton then transitions to the Await-Data state and the DATA_WAIT_TIMER is started (see section 5.6). Otherwise, the MNCP automaton transitions to the Listen

state, forwards incoming data to the receiving application (ind action), and the packet sequence is concluded.

send PT_DATA (sdt)
The MNCP automaton sends a PT_DATA packet when it receives a positive PT_ACK to a PT_NTFN or PT_DATA(more) packet, or as a retry (ACK_WAIT_TIMER expired with retry remaining).

The MNCP builds a PT_DATA packet as follows.
- Protocol Version is set to the value for this implementation.
- Packet Type is set to PT_DATA.
- Correlation Identifier is set to the value used in the PT_NTFN packet that started this packet sequence.
- Sequence Number is set to the value used in the preceding packet (retry) or that value incremented by one (otherwise).
- Data Offset is set to the starting octet position of the application-specific data to be included in this packet.
- Compressed and segmented application data (see Sections 5.7 and 5.8) is encapsulated in IE_DATA_MORE or IE_DATA_FINAL information elements. When building an IE_DATA_MORE element, the MNCP MUST fill the remainder of the PT_DATA packet with compressed data. When building an IE_DATA_FINAL element, the MNCP MUST NOT pad the data.

The PT_DATA packet is sent as a UDP packet to the source port which sent the last PT_ACK packet, the PKT_RETRY count is incremented, and the ACK_WAIT_TIMER is started (see Section 5.6). The MNCP automaton then transitions to the Data-Sent state.

start ACK_WAIT_TIMER or DATA_WAIT_TIMER (sto)
The MNCP automaton starts the ACK_WAIT_TIMER or DATA_WAIT_TIMER whenever it sends a non ACK packet, as described in Section 5.6.

send Appl Indication (ind)
The MNCP automaton supplies incoming session control information elements, application-specific information elements, and reassembled/decompressed data (if any) to the destination application just before sending a PT_ACK to an incoming PT_CMD or PT_DATA(final) packet as described under the sak action above.

send Appl Confirm (cnf)
The MNCP automaton supplies a delivery confirmation to the requesting application when it receives the final PT_ACK in a packet sequence or abandons the request. Negative confirmations include the reason why the request could not be delivered (e.g., the Ack Code value or

timeout). The MNCP automaton then transitions to the Listen state and the packet sequence is concluded.

5.6 Timers, Acknowledgment, and Retransmission

To help ensure a reliable delivery of data between a MCD and the Mobility Server, MNCP uses a stop-and-go with timeout retransmission mechanism.

Each MNCP reliable delivery packet carries a 16-bit unsigned sequence number and MUST be individually acknowledged by the receiving MNCP. The sequence number MUST start at 0 for the first packet of each packet sequence and be incremented by one for each additional packet in the flow till 65,535 and then recycled through 0 if necessary. An acknowledgment packet MUST use the same sequence number as the packet it is acknowledging.

Out of sequence data packet MUST be discarded by the receiving MNCP. As part of the action associated with the processing a PT_DATA packet that arrives out of sequence, the MNCP MUST return a PT_ACK packet containing the Sequence Number of the most recently acknowledged PT_DATA packet.

When sending a packet, the sending MNCP MUST start a retransmission timer (ACK_WAIT_TIMEOUT, default 15 seconds), during which the sending MNCP will wait for an acknowledgment from the receiving MNCP. When sending a PT_ACK to any packet other than PT_DATA (final), the receiving MNCP MUST start a DATA_WAIT_TIMER (typically, three times the ACK_WAIT_TIMER value). Expiration of this timer causes the transfer of the packet sequence to be abandoned.

If the sending MNCP does not receive an acknowledgment from the receiving MNCP within the timeout period, which has the same sequence number as the packet been sent, it should retransmit the packet up to PKT_RETRY times (default 2 retries per packet). If there is still no acknowledgment after all the retries (i.e., for an attempt of PKT_RETRY + 1 times), the sending MNCP should abort the packet sequence.

Implementation Note: In the current implementations, timers are assigned predetermined values appropriate for the wireless environment over which the MNCP operates, from a configuration file. ACK_WAIT_TIMEOUT is the same for all Service ID/Function ID combinations.

5.7 Packet Size Negotiation, Segmentation and Reassembly

A sending MNCP may propose a packet size larger than the DEFAULT_PACKET_SIZE in a NOTIFICATION REQUEST. Any value greater than DEFAULT_PACKET_SIZE but less than or equal to the maximum packet size of 2048 octets may be proposed by encoding the desired size in the IE_PKT_SIZE information element in the PT_NTFN. A receiving MNCP may

accept the proposed packet size, or it may proposed a reduced packet size. The receiving MNCP specifies the acceptable packet size (proposed or reduced) in the IE_PKT_SIZE information element when composing a PT_ACK packet in response to a PT_NTFN packet. In the absence of an explicit IE_PKT_SIZE field, the DEFAULT_PACKET_SIZE MUST be assumed.

5.7.1 Computing the payload size for PT_DATA packets

The sending MNCP subtracts the MNCP common header length (7), the length of IE_DATA_OFFSET information element (6) and the length of the IE Length and IE Type components of the IE_DATA_MORE information element (3) from the value of IE_PKT_SIZE and uses this as the PAYLOAD_SIZE.

5.7.2 Segmentation and PT_DATA Packet Composition

The sending MNCP segments application data into "n" PT_DATA packets. All but the final PT_DATA packet contain one IE_DATA_MORE information element that conveys PAYLOAD_SIZE octets, and a monotonically incremented sequence number, and a data offset.

Data compression, if selected, is performed on the application data prior to segmentation. This is required to determine the value of Compressed Message Length used in the PT_NTFN (see section 3.5.2).

Initially, a local parameter, next-data-offset, is set to zero (0), and the next-sequence-number is set equal to the value of Sequence Number sent in the PT_NTFN (zero, 0).

If no compression is used, then PAYLOAD_SIZE number of octets of uncompressed data are encoded in the Data field of the IE_DATA_MORE information element. The value of next-data-offset is encoded in the IE_DATA_OFFSET field, and next-data-offset is incremented by PAYLOAD_SIZE. If compression is used, then PAYLOAD_SIZE number of octets of compressed data are encoded in the Data field of the IE_DATA_MORE information element, the value of next-data-offset is encoded in the IE_DATA_OFFSET field, and next-data-offset is set to the octet location of the final octet of data that were encoded in the Data field. The Sequence number encoded in the common packet header is set to next-sequence number, and next-sequence-number is increased by one (1).

The PT_DATA packet is then sent as an individual transmission in a UDP packet, and a timer is initiated. If a PT_ACK is not received before the retransmission timer expires, the PT_DATA packet is retransmitted. Retransmission upon timer expiration is repeated until a maximum number of retries (PKT_RETRY +1) is exhausted, at which time the sending MNCP notifies session control of a communications failure.

Upon reception of an PT_ACK, another PT_DATA packet may be sent. If no compression is used and the value of IE_DATA_OFFSET subtracted from Original Message length is greater than PAYLOAD_SIZE, the process of composing and sending a PT_DATA packet containing an IE_DATA_MORE information element is repeated. Similarly, a PT_DATA packet containing an IE_DATA_MORE information element is composed if

compression is used and more than PAYLOAD_SIZE number of compressed octets remain to be sent. Otherwise, a final PT_DATA packet is generated.

The final PT_DATA packet contains an IE_DATA_FINAL information element, and conveys up to PAYLOAD_SIZE octets, compressed or uncompressed. The value of the IE_DATA_FINAL information element must not be padded.

5.7.3 Reassembly

The receiving MNCP processes incoming PT_DATA packets as follows. Following transmission of a positive (PT_ACK) acknowledgment to a PT_NTFN packet, the receiving MNCP sets a local parameter for next-expected-sequence-number to one (1), and awaits the arrival of a PT_DATA packet containing the same Correlation ID as encoded in the PT_NTFN packet that initiated this packet sequence and a Sequence Number equal to the parameter next-expected-sequence-number, and an IE_DATA_MORE or IE_DATA_FINAL information element.

The receiving MNCP composes and returns a PT_ACK with Sequence Number set to the value of Sequence Number from the PT_DATA packet being acknowledged, then increments next-expected-sequence-number by one (1), and awaits the arrival of the next packet in the sequence. If the sequence number in the next PT_DATA packet that arrives does not equal next-expected-sequence-number, the receiving MNCP composes and returns a PT_ACK packet with the sequence number of the last acknowledged PT_DATA packet (next-expected-sequence-number minus 1) and immediately discards the PT_DATA packet. Otherwise, the receiving MNCP processes each arriving PT_DATA packet containing an IE_DATA_MORE information element in exactly the same manner as the initial PT_DATA packet. The process repeats until a PT_DATA packet containing an IE_DATA_FINAL information element is received. The receiving MNCP composes and returns a PT_ACK as previously described.

If compression is used, the user data extracted from the IE_DATA_MORE and IE_DATA_FINAL information elements received by the MNCP are reassembled and then uncompressed. Reassembly uses the values of IE_DATA_OFFSET and PAYLOAD_SIZE to compose the original message from

the message segments delivered, then the message in its entirety is passed to the application.

Receive buffer strategies are implementation-dependent; for example, if compression is used during a transfer, the value of Compressed Message Length from the IE_MSG_LENGTH information element in the PT_NTFN may be used to allocate a buffer sufficient for the reassembly of the compressed data packet sequence, otherwise the value of Original Message Length may be used. [Note: This memo imposes no restrictions on how receive buffers are allocated in implementations.]

5.8 Data Compression

When issuing a PT_NTFN packet, the sending MNCP uses the IE_DATA_COMPRESSION field to indicate the compression method that it wishes to apply to the application data that will follow in PT_DATA packets. The IE_MSG_LENGTH Compressed Message Length field is computed

by assuming this compression method. The receiving MNCP responds to this bid in the PT_ACK packet that confirms the PT_NTFN, as follows.

To accept the proposed compression method, the responding MNCP MUST return a PT_ACK packet with IE_DATA_COMPRESSION absent and IE_ACK_CODE equal to ACK_OK.

To reject the proposed compression method, the responding MNCP MUST return a PT_ACK packet with IE_DATA_COMPRESSION set to indicate an alternative compression method (which may be NONE) and an IE_ACK_CODE equal to ACK_OOS_COMPRESS.

If the sender's proposed compression method was rejected, the sending MNCP SHOULD issue another PT_NTFN packet with an alternative compression method, repeating the above negotiation until a mutually acceptable method is agreed upon (signaled by a PT_ACK with IE_ACK_CODE equal to ACK_OK and IE_DATA_COMPRESSION absent). Note that the IE_MSG_LENGTH Compressed Message Length field is recomputed by assuming the compression method, which can either be the alternative method proposed by the receiver, or a new method proposed by the sender. If no compression method is specified, the Compressed Message Length field value MUST be the same as the value of Original Message Length.

Once a compression method has been agreed, the sending MNCP applies the negotiated method to compress the data content of PT_DATA packet IE_DATA_MORE and IE_DATA_FINAL elements sent in this MNCP packet sequence. The receiving MNCP applies the same negotiated method to

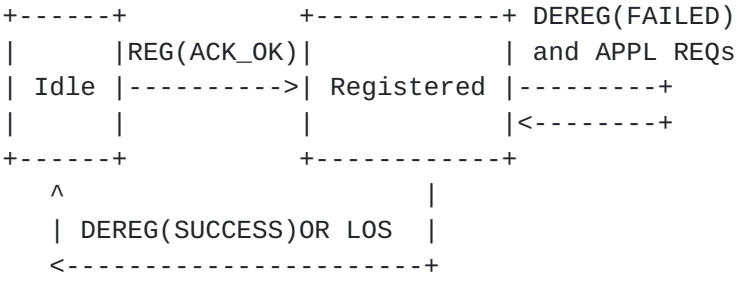
decompress the data content upon arrival, ultimately yielding the number of octets indicated by the IE_MSG_LENGTH Original Message Length field.

6. MNCP Session Control Processing

The phase diagram and state machine described in this section operates on unique Subscriber ID/Service ID pairs. When responding to any event, the session control automaton must first determine the current state associated with the Subscriber ID/Service ID pair, and then follow the course of action specified for that state.

6.1 Phase Diagram

Session control goes through two distinct phases which are specified in the following simplified diagram.



REG(ACK_OK) = Send or receive Ack Code ACK_OK to FUN_REG_REQ
or FUN_<other>
DEREG(SUCCESS)= Send/receive positive Ack accepting FUN_DEREG_REQ
PT_ACK from MCD to MS containing ACK_OOS_SVC
PT_ACK from MS to MCD containing ACK_OK
DEREG(FAILED) = Send/receive negative Ack rejecting FUN_DEREG_REQ
PT_ACK from MCD to MS containing ACK_OK
PT_ACK from MS to MCD containing any other value

APPL REQs = Send or receive FUN_<other> app-specific request
LOS = Loss of signal assumed when DEREG_REQ abandoned

Idle Phase

MNCP session control necessarily begins and ends with this phase. When a positive acknowledgment (Ack Code ACK_OK) is returned to a Registration Request (FUN_REG_REQ or FUN_<other> in the case of an implicit registration performed by the MS), session control transitions to the Registered phase.

MNCP session control returns to the Idle phase automatically after receiving or sending a successful Deregistration Request, and after timeout of a Deregistration Request, as signaled by the MNCP reliable delivery automaton.

Registered Phase

MCD MNCP session control enters this phase when receiving a positive Registration Request acknowledgment. MS MNCP session control enters this phase when it has received and processed a valid Registration request from an MCD and has responded to the request with a positive acknowledgment. MS MNCP session control also enters this phase when it has received and processed a valid service request from an MCD that has not previously (explicitly) registered with the service and has responded to the request with a positive acknowledgment.

In this phase, MNCP session control processes incoming Deregistration Requests and any other Application Request. All requests are authenticated, and service access control is verified. Application Requests are forwarded to the destination application service only if the subscriber is currently registered for that service.

The Mobility Server runs an INACTIVITY_TIMER to reaffirm the registration status on a periodic basis.

MCD MNCP session control remains in the Registered phase until it completes a successful Deregistration Request. MS MNCP session

control remains in the Registered phase until it either completes a successful Deregistration, or a Deregistration Request is abandoned due to timeout. When Deregistration is completed, the MNCP returns to the Idle phase.

6.2 State Diagram

The session control finite-state automaton is defined by events, actions and state transitions. Events include reception of application

requests and expiration of the Inactivity timer. Actions include the starting of the INACTIVITY_TIMER and transmission of requests and responses.

```
Events                                  Actions

ORR = Outgoing Registration Request     srr = send FUN_REG_REQ
ODR = Outgoing Deregistration Request   sdr = send FUN_DEREG_REQ
OAR = Outgoing Application Request      sar = send FUN_<other>
IRR = Incoming FUN_REG_REQ              au  = authenticate subscriber
IDR = Incoming FUN_DEREG_REQ            up  = update registr. status
IAR = Incoming FUN_<other>              spk = send Positive Ack
IAK = Incoming Ack                      snk = send Negative Ack
TMO = Inactivity Timeout                it  = start INACTIVITY_TIMER
```

The complete state transition table follows. States are indicated horizontally, and events are read vertically. State transitions and actions are represented in the form action/new-state. Multiple actions are separated by commas, and may continue on succeeding lines as space requires; multiple actions may be implemented in any convenient order. The state may be followed by a letter, which indicates an explanatory footnote. The dash ('-') indicates an illegal transition.

```
      | State
      |    0            1         2         3         4
Events| Listen      Reg-Sent  Dereg-Sent App-Sent Registered
------+-----------------------------------------------------
 ORR  | srr/1           -         -         -     -
 ODR  |     -           -         -         -     up,sdr/2
 OAR  |     -           -         -         -     sar/3
 IRR  | au,snk/0 or     -         -         -     -
      | au,up,it,spk/4
 IDR  |     -       snk/1      snk/2      snk/3   au,snk/4 or
      |                                           au,up,spk/0
 IAR  |     -           -         -         -     au,it,up,spk/4 or
      |                                           au,it,spk/4 or
      |                                           au,it,snk/4
 IAK  |     -       up/4 or   up,it/4 or    4     -
      |                 0         0
 TMO  |     -           -         -         -     up,sdr/2
```

The INACTIVITY_TIMER runs only at the Mobility Server.

6.3 States

Following is a more detailed description of each automaton state.

Listen
  The session control automaton begins and ends in this state, awaiting either a locally-initiated (outgoing) Registration Request, or receipt of an incoming FUN_REG_REQ indication from MNCP reliable delivery.

Registration-Request-Sent
  The session control automaton transitions to this state when sending a FUN_REG_REQ. The events expected to occur while in this state are receipt of an Ack Code from MNCP reliable transfer, indicating the result of the request or timeout, or receipt of a Deregister request form the MS.

Deregistration-Request-Sent
  The session control automaton transitions to this state when sending a FUN_DEREG_REQ. Events expected to occur while in this state are receipt of an Ack Code from MNCP reliable transfer or receipt of an incoming FUN_DEREG_REQ.

Application-Request-Sent
  The session control automaton transitions to this state when sending a FUN_REG_REQ. Events expected to occur while in this state are receipt of an Ack Code from MNCP reliable transfer or receipt of an incoming FUN_DEREG_REQ, or receipt of a Deregister request form the MS.

Registered
  The session control automaton transitions to this state when returning a positive Ack for an incoming FUN_REG_REQ (explicit registration), returning a positive Ack for an incoming FUN_<other> (implicit request), receiving a positive Ack for an outgoing FUN_REG_REQ, receiving a positive Ack for an outgoing FUN_<other>, or receiving a negative Ack for an outgoing FUN_DEREG_REQ. Events expected to occur while in this state are outgoing Deregistration or Application Requests, incoming FUN_DEREG_REQ or FUN_<other> packets, or expiration of the INACTIVITY_TIMER.

6.4 Events

Transitions and actions in the automaton are caused by events.

Outgoing Registration Request (ORR)
  This event occurs when a Registration Request is initiated by the MCD, causing the session control automaton to send a FUN_REG_REQ (srr action). This event MAY also be initiated implicitly by the MCD session control implementation (e.g., when the first Application-specific Request for a give service is initiated).

Outgoing Deregistration Request (ODR)
  This event occurs when a Deregistration Request is initiated by the MCD, causing the session control automaton to send a FUN_DEREG_REQ (sdr action). This event MAY also be initiated by the MS session

control implementation (as a consequence of the expiry of the INACTIVITY_TIMER).

Outgoing Application Request (OAR)
This event occurs when an Application Request is initiated by the MCD, causing the session control automaton to send a FUN_<other> (sar action).

Incoming FUN_REG_REQ (IRR)
This event occurs when a remotely-initiated FUN_REG_REQ is received by the Mobility Server. The session control automaton attempts to authenticate the subscriber and verify the subscriber is permitted access to the service indicated in the request (au action) and returns the result to MNCP reliable delivery as an Ack Code (snk or spk action). When authentication is successful, the automaton also updates the subscriber's registration status (up action) and starts the INACTIVITY_TIMER (it action).

Incoming FUN_DEREG_REQ (IDR)
This event occurs when a remotely-initiated FUN_DEREG_REQ is received by either the MCD or Mobility Server. The session control automaton attempts to authenticate the subscriber (au action) and returns the result to MNCP reliable delivery as an Ack Code (snk or spk action). When authentication is successful, the automaton also updates the subscriber's registration status (up action).

Incoming FUN_<other> (IAR)
This event occurs when any other remotely-initiated request (FUN_<other>) is received by either the Mobility Server. The session control automaton attempts to authenticate the subscriber and verify the subscriber is permitted access to the service indicated in the request (au action), and returns the result to MNCP reliable delivery as an Ack Code (snk or spk action).

Incoming Acknowledgment (IAK)
This event occurs when a response is received from MNCP reliable delivery, indicating success or failure of an earlier request as an Ack Code. When receiving a Registration or Deregistration Request acknowledgment, the automaton updates the subscriber's registration status (up action). When the Mobility Server receives an ACK_OK in a Deregistration Request acknowledgment, indicating the subscriber

should be (re)registered for the service, the MS (re)starts the INACTIVITY_TIMER (it action).

Inactivity Timeout (TMO)
This event occurs when the INACTIVITY_TIMER started by the Mobility Server's session control automaton expires. Subsequent processing assumes that the MCD has become unavailable since last contact (e.g., out of range, turned off, disabled). The automaton updates the subscriber's registration status (up action) and sends a FUN_DEREG_REQ (sdr action).

6.5 Actions

Actions in the automaton are caused by events and typically indicate the transmission of packets and/or the (re)starting of the Inactivity timer.

send FUN_REG_REQ (srr)
The MCD sends a FUN_REG_REQ packet as a result of a locally-initiated Registration Request involving a Subscriber/Service combination that is not currently registered (i.e., the automaton is in the Listen state). Session Control fields (Service ID, Function ID, Subscriber ID, Subscriber Password) are supplied to the MNCP reliable delivery automaton for inclusion in a PT_CMD packet. Function ID is set to FUN_REG_REQ; all other values are obtained from the MCD subscriber/application. The session control automaton then transitions to the Registration-Request-Sent state.

send FUN_DEREG_REQ (sdr)
The MCD sends a FUN_DEREG_REQ packet as a result of a locally-initiated Deregistration Request (implicit or explicit) involving a Subscriber/Service combination that is currently registered (i.e., the automaton is in the Registered state). The Mobility Server also sends a FUN_DEREG_REQ packet when the INACTIVITY_TIMER expires. Session Control fields (Service ID, Function ID, Subscriber ID, Subscriber Password) are supplied to the MNCP reliable delivery automaton for inclusion in a PT_CMD packet. Function ID is set to FUN_DEREG_REQ; all other values are obtained from the subscriber/application or the registration status table. The session control automaton marks the subscriber as deregistered for this service (up action) and then transitions to the Deregistration-Request-Sent state.

send FUN_<other> (sar)

Either the Mobility Server or the MCD sends a FUN_<other> packet as a result of a locally-initiated Application Request involving a Subscriber/Service combination that is currently registered (i.e., the automaton is in the Registered state), or the MCD sends a FUN_<other> packet as a result of a locally-initiated Application Request involving a Subscriber/Service combination that has not been explicitly registered (i.e., the MS is expected to implicitly register the application at the MCD). Session Control fields provided by the application (Service ID, Function ID, Subscriber ID, Subscriber Password) are supplied to the MNCP reliable delivery automaton for inclusion in a PT_CMD or PT_NTFN packet. The session control automaton then transitions to the Application-Request-Sent state.

An Application Request initiated when the automaton is in the Listen state MAY cause the MCD to attempt implicit registration before the request is further processed. Otherwise, such a request MUST be rejected using the Ack Code value ACK_OOS_SVC and the automaton remains in the Listen state.

authenticate subscriber (au)

The automaton attempts to authenticate the Subscriber ID and Password and verifies the subscriber's current registration status and permission to access the specified Service ID and Function ID.

- If Subscriber ID is not found, return the Ack Code ACK_ERR_SID.
- Else if the Subscriber Password is invalid, return the Ack Code ACK_ERR_PWD.
- Else if the Subscriber does not have permission to access this Service and Function, return the Ack Code ACK_OOS_SID.
- Else if the Service is not currently available (e.g., the application process bound to this Service ID is not running), return ACK_OOS_SVC.
- Else return ACK_OK.

The method by which the MNCP determines whether a local application service is running is not specified by this memo. Refer to Section 7 for additional discussion of Security. Refer to the spk and snk actions for Ack Code processing.

update Registration Status (up)

The automaton updates the subscriber's current registration status for each specified service as follows.

When the MNCP:	Status is set to:
Sends positive FUN_REG_REQ Ack Code	Registered
Receives positive FUN_REG_REQ Ack Code	Registered
Receives positive FUN_<other> Ack Code	Registered (implicit)
Sends FUN_DEREG_REQ	Deregistered (implicit)
Sends positive FUN_DEREG_REQ Ack Code (MCD sends ACK_OOS_SVC, MS sends ACK_OK)	Deregistered
Receives positive FUN_DEREG_REQ Ack Code (MCD rcvs ACK_OK, MS rcvs ACK_OOS_SVC)	Deregistered
Sends negative FUN_DEREG_REQ Ack Code	(Still)Registered

(MCD sends ACK_OK, MS sends ACK_other)
Receives negative FUN_DEREG_REQ Ack Code (Re)Registered
(MCD rcvs ACK_other, MS rcvs ACK_OK)

Storage methods and internal representation of subscriber information and registration status are not specified by this memo.

send Positive Ack Code (spk)
Successful authentication of any incoming request forwarded to session control is indicated by returning a positive ACK code to the MNCP reliable delivery automaton for inclusion in a PT_ACK packet. A positive Ack Code is ACK_OK in all cases except when an MCD responds to a FUN_DEREG_REQ; in this case, it is ACK_OOS_SVC. The session control automaton changes state when acknowledging initial Application Requests (FUN_<other> packets) that cause implicit registration. The automaton transitions to the logical next state: Registered after a FUN_REG_REQ, FUN_<other>, or Listen after a FUN_DEREG_REQ.

send Negative Ack Code (snk)
Failed authentication of any incoming request forwarded to session control is indicated by returning a negative ACK code to the MNCP reliable delivery automaton for inclusion in a PT_ACK packet. A negative ACK code is one of {ACK_ERR_SID, ACK_ERR_PWD, ACK_OOS_SID, or ACK_OOS_SVC}, except in the case where an MCD rejects a FUN_DEREG_REQ, in which case the ACK code is ACK_OK. The session control automaton does not change state when acknowledging Application Requests (FUN_<other> packets) or failed Registration Requests, but transitions back to the Registered state after a failed Deregistration Request (i.e., MS requests deregistration after inactivity time expires, but MCD indicates the application is still active).

start INACTIVITY_TIMER (it)
The Mobility Server's session control automaton (re)starts an INACTIVITY_TIMER whenever it receives a FUN_REG_REQ, FUN_<other>, or failed FUN_DEREG_REQ Ack from the MCD. This timer allows the Mobility Server to refresh the registration status associated with an MCD on a periodic basis in the absence of other traffic. This timer does not apply to the MCD's session control automaton.

7. Security Considerations

The only form of security provided in this protocol is a validation (weak authentication) scheme based on clear text subscriber identification and password, so it is vulnerable to several known forms of attacks on clear text, against which only limited defenses can be taken (password aging/expiration, use of one-time long, system-generated passwords, etc.). Only subscriber validation is performed (the subscriber has no mechanism to determine the authenticity of a mobility server).

[NOTE: A means of negotiating or selecting a strong authentication method and (additionally, optionally) a method for providing data integrity and confidentiality at the session control level of MNCP is under investigation.]

Since MNCP operates over UDP/IP, it may be appropriate to make use of security services offered by other layers. For example, an IP Authentication Header can be used to provide integrity and authentication without confidentiality to IP datagrams [6], whereas an IP Encapsulating Security Payload (ESP) can be used to provide integrity, authentication, and confidentiality to IP datagrams (see also [7]).

For other applications, it may be appropriate to adopt/adapt application-specific security services. For example, Mobile Messaging application, having already adapted POP3 [8] and SMTP [9], could also adapt PGP [10] to provide non-repudiation of sender and data privacy through encryption.

8. References

[1]Postel, J., "User Datagram Protocol," RFC 768, USC/Information Sciences Institute, 28 August 2880.

[2] Reynolds, S. and Postel, J., "ASSIGNED NUMBERS", RFC 1700, 20 October 1994.

[3] Friend, R. and Simpson, W., "PPP Stac LZS Compression Protocol", RFC 1974, USC/Information Sciences Institute, 13 August 2896.

[4] Malkin, G., "Internet Users' Glossary", RFC 1983, USC/Information Sciences Institute, 16 August 2896.

[5] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", RFC 2119, USC/Information Sciences Institute, 26 March 1997.

[6] Atkinson, R., "IP Authentication Header", RFC 1826, USC/Information Sciences Institute, 9 August 2895.

[7] Atkinson, R., "Security Architecture for the Internet Protocol", RFC 1825, USC/Information Sciences Institute, 9 August 2895.

Exchange" Formats", RFC 1991, , USC/Information Sciences Institute, 16 August 2896

[9] S Myers, J., M. Rose, "Post Office Protocol - Version 3", RFC 1939, USC/Information Sciences Institute, 14 May 1996

[10] Postel, J., "Simple Mail Transfer Protocol", RFC 821, USC/Information Sciences Institute, January 8 1982

9. Authors' Addresses

Dave Piscitello
Core Competence
1620 Tuckerstown Road
Dresher, PA 19025
(215) 830-0692
dave@corecom.com

Lisa Phifer
Core Competence
1620 Tuckerstown Road
Dresher, PA 19025
(215) 830-0692
lisa@corecom.com

Richard Hovey
Bellcore
445 South Street, MRE-2N264
Morristown, NJ 07960-6438
(201)829-4176
hovey@bellcore.com

Yangwei Wang
Bellcore
331 Newman Springs Rd, NVC-3C221
Red Bank, NJ 07701
(908)758-5107
ywang@cc.bellcore.com

Appendix A. HDML Transactions using MNCP

The design of the Mobile Network Computing Protocol (MNCP) makes it possible to implement a wide range of services efficiently and reliably using low-bandwidth, high-latency wireless links. In this Appendix, we describe the implementation of a Web browsing and information push service based on the Handheld Device Markup Language (HDML).

HDML has been proposed as a means of bringing interactive browsing services to devices with limited processing, storage, display and input capabilities. The language is based on the concept of decks (the basic unit transported) and cards (the basic unit displayed). The language is a replacement for HTML for portable devices such as screen telephones and two-way pagers, and makes use of the infrastructure of HTTP connections and Web servers that has been created for HTML browsers. (See http://www.w3.org/pub/WWW/TR/NOTE-Submission-HDML.html for a copy of the HDML proposal)

An HDML browser on an MNCP-capable client device begins by registering with the Mobility server. The client registers by submitting a request with the Function ID set to 1, and the Service ID set to the pre-defined HDML service (assume 85 is assigned to HDML). During the registration process, the Mobility server validates the user, based on the submitted Subscriber ID and password, ensuring both that the subscriber is registered with the server and authorized to use the requested service. Once registered, the client device can perform interactive browsing of HDML sites and receive unsolicited (pushed) messages encoded in HDML.

Browsing of HDML sites is initiated by the subscriber, pointing the browser to the desired URL. The HDML browser formulates a HTTP request (e.g., GET or POST) of the specified URL, and submits this to the MNCP agent code on the client device. The MNCP code then formulates a single packet or multi-packet request to a pre-defined Service ID (i.e., 85)

on the Mobility server. The MNC protocol handles reliable delivery of the request to the Mobility server, invisible to the HDML browser.

The Mobility server routes incoming messages to code modules based on the Service ID. Any message with a Service ID of 85 are routed to the HDML service module. The HDML service module then extracts the HTTP request from the MNCP message. The HTTP request could be implemented by using a Function ID to correspond to each HTTP function (i.e., Function ID 2 maps to GET, Function ID 3 maps to PUT) or the HTTP request as formulated by the client could be included as the body of the message. The HDML service module then forwards the HTTP request to the HDML information server. When the Mobility server receives the response from the HDML information server, it encodes the response with the previously negotiated compression algorithm and delivers it to the client device. The subscriber could then continue browsing in a similar fashion, based on the contents of the HDML deck that was returned.

The process of `pushing' a message to the HDML browser is implemented from the server side. An external server begins the process by formulating an HDML deck to be delivered to a particular HDML service subscriber. Alternatively, a person may wish to send an email- or page-type notice to the subscriber. The information server or the individual must deliver the message to the Mobility server, using the interface specified by the Mobility server. (HTTP connections or e-mail messages are two possibilities.) When the Mobility server receives such a message, it first determines if the subscriber to whom the message is addressed is registered. If the subscriber is not registered for HDML service, the message might be returned or placed in a message store. If the subscriber is currently registered, the Mobility server routes the message to the HDML service module. The HDML service module takes the incoming message and formats it for delivery. This formatting might include converting a plain-text message to HDML, as well as compression and segmentation for MNCP delivery. The HDML service assigns the proper Service ID (85) and a pre-defined Function ID that corresponds to pushed messages. In fact, multiple Function IDs could be assigned to pushed messages, with different values indicating different levels of priority, for example. Using single- or multiple-packet reliable delivery, the Mobility server then forwards the message to the HDML client. The browser code receives the HDML deck, interprets the format for proper display and informs the user in an appropriate fashion. The user can then interact

with the delivered message in standard browsing mode.

When the subscriber has completed browsing and/or no longer wishes to receive pushed messages, terminating the browser will perform a de-registration operation, separating the subscriber from the Mobility server. The de-registration message again uses the HDML-assigned Service ID (85) and the Function ID of 0.

A variety of services can be implemented using HDML. Currently, public services such as phone-number searches, news reports and weather forecasts are available. Users of such services would not generally benefit from the overhead of sophisticated security measures. However, HDML could be used in intranet-like applications, such as salespeople querying market data, which would require encryption and authentication. To properly protect the data transmitted in such applications, the security measures must protect the entire path from client to server, not just the wireless portion of the link. End-to-end encryption and authentication protocols could be built into HDML browsers and HDML information servers to protect sensitive data.

Figure A-1 depicts the MNC architecture as used to support HDML transactions. Figure A-2 illustrates the message flows.

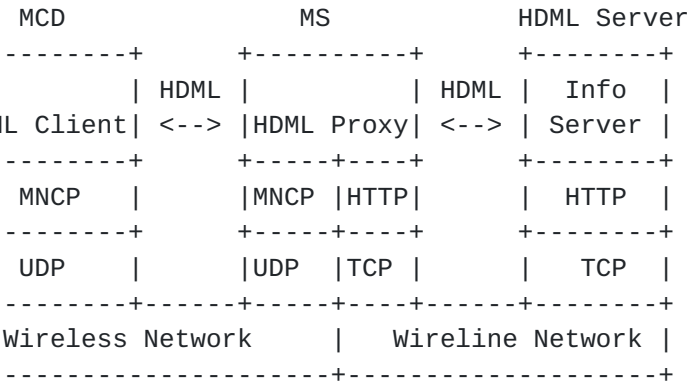
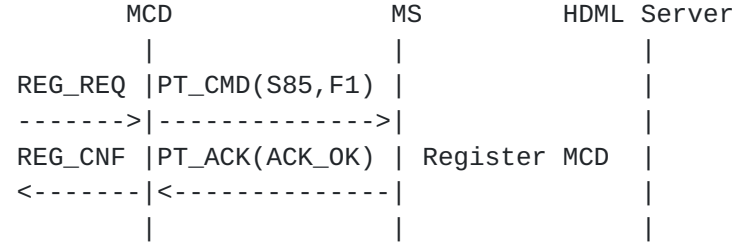
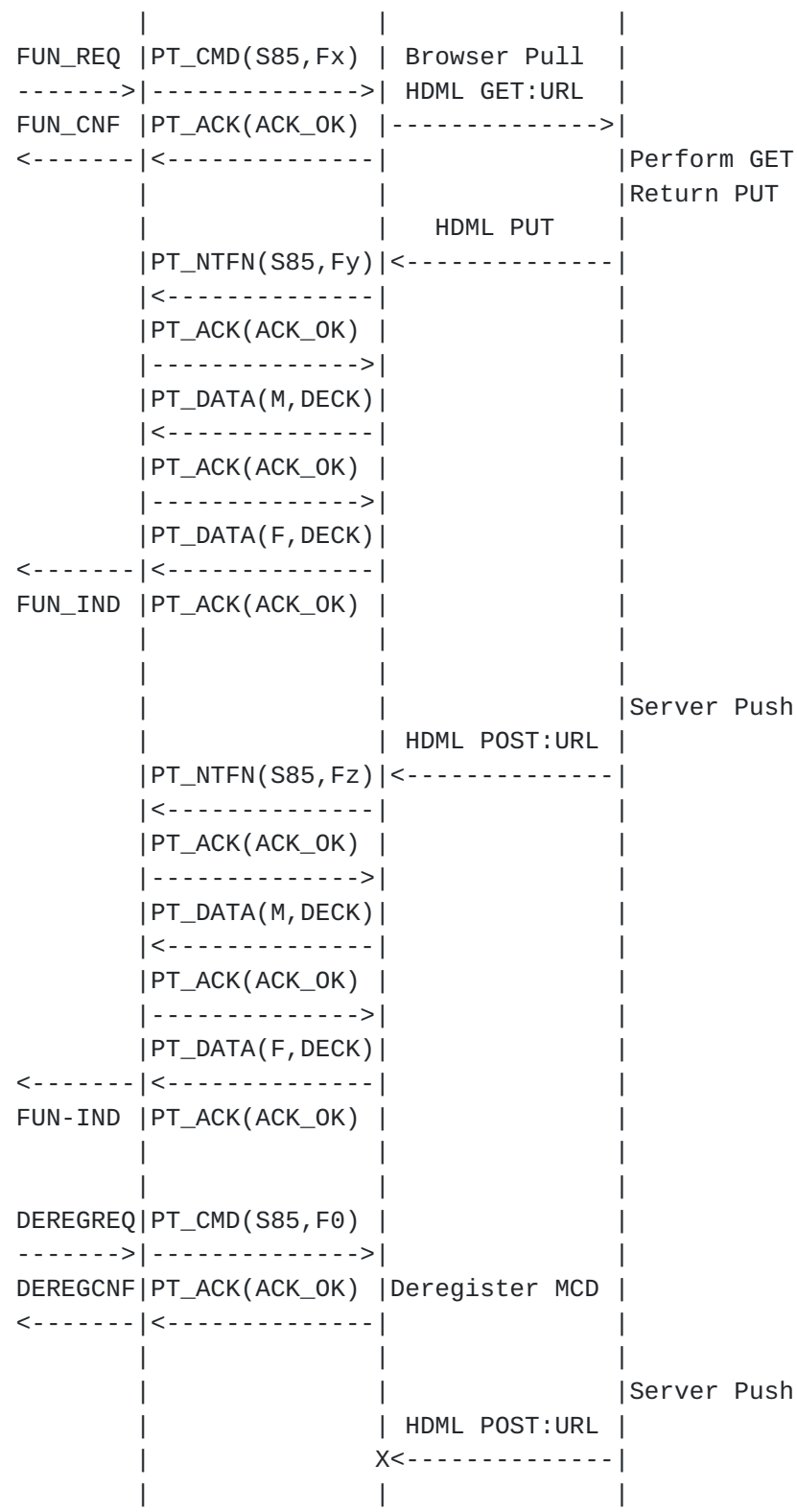


Figure A-1.


MCD
MS
HDML Server
REG_REQ
PT_CMD(S85,F1)
REG_CNF
PT_ACK(ACK_OK)
Register MCD


FUN_REQ
PT_CMD(S85,Fx)
Browser Pull
HDML GET:URL
FUN_CNF
PT_ACK(ACK_OK)
Perform GET
Return PUT
HDML PUT
PT_NTFN(S85,Fy)
PT_ACK(ACK_OK)
PT_DATA(M,DECK)
PT_ACK(ACK_OK)
PT_DATA(F,DECK)
FUN_IND
PT_ACK(ACK_OK)
Server Push
HDML POST:URL
PT_NTFN(S85,Fz)
PT_ACK(ACK_OK)
PT_DATA(M,DECK)
PT_ACK(ACK_OK)
PT_DATA(F,DECK)
FUN-IND
PT_ACK(ACK_OK)
DEREGREQ
PT_CMD(S85,F0)
DEREGCNF
PT_ACK(ACK_OK)
Deregister MCD
Server Push
HDML POST:URL

Key: S85 = Service ID of HDML-based service
F1 = Function ID of FUN_REG_REQ
F0 = Function ID of FUN_DEREG_REQ
Fx,y,z = Function IDs of FUN_<other>_REQs
M,DECK = IE_DATA_MORE containing segment of HDML deck
F,DECK = IE_DATA_FINAL containing last segment of HDML deck

Note: This flow illustrates explicit registration. For implicit registration, omit first MNCP packet sequence; MS registers MCD upon receipt of FUN_<other>_REQ instead.

Figure A-2.

Appendix B. Future Protocol Extensions

Several extensions are under consideration for the MNCP.

A reliable delivery that uses a sliding window mechanism to improve performance. The objective is to extend the current positive acknowledgment of packets with retransmission by allowing more than one packet to be transmitted before waiting for acknowledgment from the receiver. Full or partial window credit indications would accompany cumulative acknowledgments returned by the receiver, and selective acknowledgments would be a desirable extension as well

A means of providing data synchronization at the session control level that persists beyond the possibly abrupt termination or interruption of an underlying connection is desirable. Such a mechanism would allow a mobile computing device to continue a transfer from a commonly agreed upon starting point in an octet stream rather than from the beginning of the stream.

It would be desirable to be able to extend the existing "per reliable transfer" authentication model in MNCP to support strong authentication whether applications are explicitly or implicitly registered. It would also be useful to allow the MCD to verify the authenticity of a Mobility Server.

One method under consideration is to identify an authentication method and encode the authentication information appropriate for that method in the same registration request or data request packet. The initiator (MCD or MS) chooses a method, performs whatever computation is required to generate the strong authentication information for that method, and then sends the packet. The initiator decides what security is appropriate. A new IE identifying authentication-method can be sent in the clear, with the remainder of the PT_NTFN (PT_CMD) (including authentication information) encrypted as per the indicated security method.