FFI RAPPORT

SPECIFICATION OF A SOCKET INTERFACE FOR A GENERIC TACTICAL PROTOCOL SOLUTION

EGGEN Anders

FFI/RAPPORT-2004/01162

SPECIFICATION OF A SOCKET INTERFACE FOR A GENERIC TACTICAL PROTOCOL SOLUTION

EGGEN Anders

FFI/RAPPORT-2004/01162

FORSVARETS FORSKNINGSINSTITUTT
Norwegian Defence Research Establishment
P O Box 25, NO-2027 Kjeller, Norway

FORSVARETS FORSKNINGSINSTITUTT (FFI)
Norwegian Defence Research Establishment

P O BOX 25
N0-2027 KJELLER, NORWAY

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE
(when data entered)

REPORT DOCUMENTATION PAGE

1) PUBL/REPORT NUMBER	2) SECURITY CLASSIFICATION	3) NUMBER OF PAGES
FFI/RAPPORT-2004/01162	UNCLASSIFIED	24
1a) PROJECT REFERENCE	**2a) DECLASSIFICATION/DOWNGRADING SCHEDULE**	
FFI-II/869/110	-	

4) TITLE

SPECIFICATION OF A SOCKET INTERFACE FOR A GENERIC TACTICAL PROTOCOL SOLUTION

5) NAMES OF AUTHOR(S) IN FULL (surname first)

EGGEN Anders

6) DISTRIBUTION STATEMENT

Approved for public release. Distribution unlimited. (Offentlig tilgjengelig)

7) INDEXING TERMS

IN ENGLISH:	IN NORWEGIAN:
a) Tactical	a) Taktisk
b) Protocol	b) Protokoll
c) P-Mul	c) P-Mul
d) ACP 142	d) ACP 142
e) JAVA Sockets	e) JAVA Sockets

THESAURUS REFERENCE:

8) ABSTRACT

This report gives a specification of a JAVA™ socket API (Application Protocol Interface) on top of a generic protocol solution developed for use over communication systems with low data rate. The solution may be used for unicast or multicast communication, and will handle recipients in EMCON (radio silence). The JAVA™ API described is generic in the sense that it provides an API identical to the interface of the java.net socket communication *package*. Therefore, the applications normally using the java.net communication *package*, do not need to be modified in order to use this protocol solution. The difference is however that instead of mapping the socket communication to the TCP/IP or UDP/IP protocols as is done in java.net, the socket communication is mapped down to a different protocol stack consisting of a tactical adaptation sublayer, ACP 142 (P-Mul), UDP and IP. All of the protocols are described in the report.

This report is written in the form of a specification aimed at vendors for implementation.

9) DATE	AUTHORIZED BY This page only	POSITION
2004-03-26	Vidar S. Andersen	Director

ISBN 82-464-0836-4

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE
(when data entered)

CONTENTS

Page

SPECIFICATION OF A SOCKET INTERFACE FOR A GENERIC TACTICAL PROTOCOL SOLUTION

1 INTRODUCTION

This report gives a specification of a JAVA™ socket API over a generic tactical protocol profile, which may be used by most applications in order to transfer information over communications systems with variable communications quality and low data rate.

Most applications communicate using TCP or UDP "Sockets", which API is provided by the operating system or programming language used.

TCP is a connection oriented transport protocol, which is not ideal for use over communications systems with high error rates and low data-rates. First, TCP is designed for relatively high data-rate networks (in this context > 20 kbps), where the loss of packets is caused by congestion and not errors. Second, TCP is connection oriented and has a connection establishment and termination phase using three-way-handshake. This is bandwidth consuming for communication systems with low data rate. Third, a connection oriented protocol stack is not suited for multicasting and handling of EMCON recipients.

UDP is a connectionless transport protocol, which in addition to the functionality of the IP protocol, only adds socket addressing and checksum calculation (over both header and data). UDP does not provide any guarantee to the transfer of the data, but leaves functionality like acknowledgement and retransmissions to the application. If the delivery of the data is crucial, UDP may not be used without any other protocol in addition to ensure the delivery of the data. In this specification the ACP 142 (P_Mul) protocol is used to i.a. ensure the delivery of the data to the recipient side.

This specification describes how a JAVA socket interface is to be added to this protocol solution in order to make it more generic and usable for applications using JAVA™ socket communications defined by the "java.net" package (ref. (10)).

The aim of the solution is to be able to provide the applications with the same socket interface they are used to see, but replace the protocol stack beneath with a reliable connection less protocol profile, which may be used for unicast, multicast and handling of EMCON recipients.

The solution is a generalization of the author's protocol proposals for Tactical MMHS (ref. (1)) and tactical Directory DISP (ref. (9)). The tactical MMHS have been successfully tested by the NDRE and developed as a commercial product by i.a. THALES Communications.

1.1 Document Conventions

This specification uses the same keywords as specified in (ref. (8)) for defining the significance of each particular requirement. These words are:

MUST
This word, or the terms "REQUIRED" or "SHALL", mean that the definition is an absolute requirement of the specification.

MUST NOT
This phrase, or the phrase "SHALL NOT", means that the definition is an absolute prohibition of the specification.

SHOULD NOT
This phrase, or the phrase "NOT RECOMMENDED" means that there may exist valid reasons in particular circumstances when the particular behaviour is acceptable or even useful, but the full implications should be understood and the case carefully weighed before implementing any behaviour described with this label.

MAY
This word, or the adjective “OPTIONAL”, means that an item is truly optional. One vendor may choose to include the item because a particular marketplace requires it or because the vendor feels that it enhances the product while another vendor may omit the same item. An implementation, which does not include a particular option, MUST be prepared to interoperate with another implementation, which does include the option, though perhaps with reduced functionality. In the same way, an implementation, which does include a particular option, MUST be prepared to interoperate with another implementation, which does not include the option (except, of course, for the feature the option provides.)

1.2 Elements For Layer-to-Layer Communication

1.2.1 Definition of Service Primitives and Parameters

Communication between layers is accomplished by means of service primitives. Service primitives represent, in an abstract way, the logical exchange of information and control between the adjacent layers. Service primitives consist of commands and their respective responses associated with the services requested of another layer. The general syntax of a primitive is:

X-Service.type (Parameters)
where X designates the layer providing the service. Service primitives are not the same as an application programming interface (API) and are not meant to imply any specific method of implementing an API. Service primitives are an abstract means of illustrating the services provided by the protocol layer to the layer above. The mapping of these concepts to a real API and the semantics associated with a real API is an implementation issue and is beyond the scope of this specification.

1.2.2 Time Sequence Charts

The behaviour of service primitives is illustrated using time sequence charts, which are described in (ref. 11).

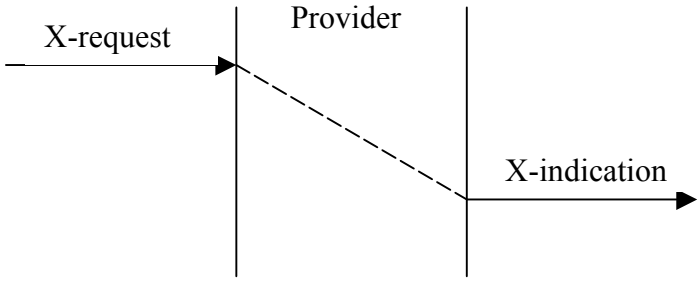

Figure 1.1 A Non-confirmed Service

Figure 1.1 illustrates a simple non-confirmed service, which is invoked using a request primitive and results in an indication primitive in the peer. The dashed line represents propagation through the provider over a period of time indicated by the vertical difference between the two arrows representing the primitives.

1.2.3 Primitive Types

The primitives types defined in this specification are:

Type	Abbreviation	Description
request	req	Used when a higher layer is requesting a service from the next lower layer
indication	ind	A layer providing a service uses this primitive type to notify the next higher layer of activities related to the request primitive type of the peer (such as the invocation of the request primitive) or to the provider of the service (such as a protocol generated event)
response	res	A layer uses the response primitive type to acknowledge receipt of the indication primitive type from the next lower layer
confirm	cnf	The layer providing the requested service uses the confirm primitive type to report that the activity has been completed successfully

1.2.4 Service Parameter Tables

The service primitives are defined using tables indicating which parameters are possible and how they are used with the different primitive types. For example, a simple confirmed primitive might be defined using the following:

Primitive **Parameter**	**X-primitive**			
	req	***ind***	***res***	***cnf***
Parameter 1	M	M(=)		
Parameter 2			O	C(=)

If some primitive type is not possible, the column for it will be omitted. The entries used in the primitive type columns are defined in the following table:

M	Presence of the parameter is mandatory – it MUST be present
C	Presence of the parameter is conditional depending on values of other parameters
O	Presence of the parameter is a user option – it MAY be omitted
P	Presence of the parameter is a service provider option – an implementation MAY not provide it
-	The parameter is absent
*	Presence of the parameter is determined by the lower layer protocol
(=)	The value of the parameter is identical to the value of the corresponding parameter of the preceding service primitive

In the example table above, *Parameter 1* is always present in *X-primitive.request* and corresponding *X-primitive. indicatio*n. *Parameter 2* MAY be specified in *X-primitive.response* and in that case it MUST be present and have the equivalent value also in the corresponding *X-primitive.confir*m; otherwise, it MUST NOT be present.

2 PROTOCOL ARCHITECTURE OVERVIEW

Figure 2.1 shows the generic tactical protocol architecture. Each of the layers shown in the figure is described in the following sections.

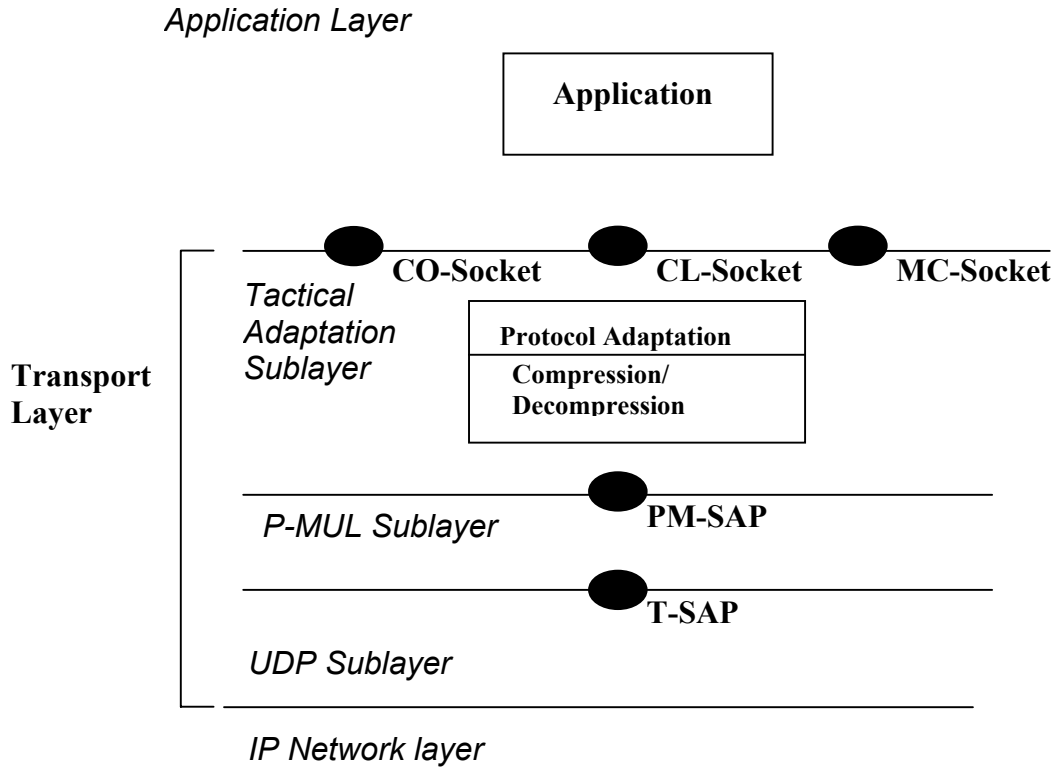

Figure 2.1 The Generic Tactical Protocol Architecture

3 THE APPLICATION LAYER

The Application layer consists of any application, which may communicate using either of the JAVA™ sockets defined in the java.net package: *Socket, ServerSocket and SocketImpl* (TCP), *DatagramSocket, DatagramSocketImpl, DatagramPacket* (UDP), or *MulticastSocket* (UDP).

4 THE TACTICAL ADAPTATION SUB-LAYER

The Tactical Adaptation sublayer is required in order to provide the expected JAVA™ socket interfaces to the applications, and to perform the adaptation of these services to the P_Mul sublayer service interface. In addition the Tactical Adaptation Sublayer implements the "Tactical Adaptation Sublayer Data Type" protocol and performs operations to increase the throughput, like compression/decompression. The main concern of this layer is to make it possible to use the standard applications without modification, while reducing the bandwidth usage to a minimum using an efficient reliable connectionless protocol stack.

We have chosen to use a connectionless protocol stack consisting of the "Tactical Adaptation Sublayer Data Type" protocol, ACP 142 protocol, the UDP protocol and the IP protocol. This protocol stack will be used for all the socket interfaces shown in figure 3.1. The P-Mul protocol described in ACP 142, will replace the TCP protocol (but still providing a TCP JAVA™ socket interface). The P-Mul protocol uses the UDP/IP protocol and adds reliability over UDP, in that it fragments the data into smaller PDUs, includes sequence numbers and checksums and ensures retransmission of lost PDUs using the selective repeat mechanism.

To make this protocol solution transparent for the applications, we need this adaptation layer in order to map the JAVA™ socket API to the actual functionality of this connectionless protocol stack.

4.1 Functionality to Increase the Throughput Over Low Data-rate Connections

The data throughput is increased through different means:

1) Use of compression: All the data from the application is compressed (see section 4.4).
2) Both the CO and CL socket service is mapped onto a connectionless protocol stack in order to avoid the overhead of connection establishment and termination at each layer.

4.2 The Tactical Adaptation Sublayer API

The API provided by the Tactical Adaptation Sublayer, SHALL provide three socket interfaces, one connection oriented (CO), one connectionless (CL), and one connectionless Multicast interface.

Because the protocol stack used is different from the standard JAVA™ communications protocol stacks, it requires the services provided by the Tactical Adaptation Sublayer API to simulate the expected functionality in order for the service API to look the same.

The implementation of this specification SHALL provide an API (or *package*) identical to the JAVA™ socket interface (API) defined by java.net *package*. The rational is that by making the APIs identical, it is not necessary to make any modifications to the applications themselves. Instead of importing the java.net *package*, the applications can import this new *package*, which provide the same interface, but uses a different protocol stack. There are many ways to use the socket communication API in JAVA™ and the different applications will utilize the API differently. This specification don't instruct on how to implement each of the *fields, methods, constructors* and *classes* of the API, but leave the implementation decisions to the vendors.

4.2.1 Connection Oriented Socket Interface (API)

The CO-Socket interface SHALL be identical to the JAVA™ socket interface (API) defined by the classes *Socket, ServerSocket and SocketImpl*, which are used to communicate using the TCP/IP protocols, and which is defined by the java.net *package*. There SHALL be no difference in the way the applications communicate with the CO-Socket Interface of the Tactical Adaptation Sub-Layer and this JAVA™ *Socket* interface. This means that all of the *classes, fields*, *methods* and *constructors* need to be present and simulated, as well as the responses the application expects to see when invoking them. The API SHALL be implemented by using the same names and the same parameters/return values. The functionality of the services will however be different because they are simulated.

The Tactical adaptation sublayer SHALL simulate the connection establishment and termination phase of the TCP protocol and only transfer the data of the data transmission phase including a one way termination (shown in section 4.1.3).

The data transmission phase of the CO-Socket interface SHALL be mapped down to the PM-DATA.req primitive of the P_Mul sublayer and use the "unicast" service of the ACP 142 protocol (see ref. 3 for details).

4.2.2 Connection Less Socket Interface

The CL-Socket interface SHALL be identical to the JAVA™ *DatagramSocket* interface (API) defined by the *ServerSocket*, *DatagramSocket, DatagramSocketImpl* and *DatagramPacket* of the java.net *package*. There SHALL be no difference in the way the applications communicate with the CL-Socket Interface of the Tactical Adaptation Sub-Layer and the *Datagram* JAVA™ interface. This means that all of the *classes, fields*, *methods* and *constructors* need to be present and simulated, as well as the responses the application expects to see when invoking them. This functionality SHOULD be implemented by using the same names and the same parameters/return values. The functionality of the services will however be different because of the simulation of the services.

The data transmission phase of the CO-Socket interface SHALL be mapped down to the PM-DATA.req primitive of the P_Mul sublayer and use the "unicast" service of the ACP 142 protocol (see ref. 3 for details).

4.2.3 Multicast Socket interface

The MC-Socket interface SHALL be identical to the JAVA™ *MulticastSocket* interface (API) defined in java.net *package* and the corresponding server side. There SHALL be no difference in the way the applications communicate with the MC-Socket Interface of the Tactical Adaptation Sub-Layer and the JAVA™ *MulticastSocket* interface. This means that all of the *classes, fields*, *methods* and *constructors* need to be present and simulated, as well as the responses the application expects to see when invoking them. This functionality SHOULD be implemented by using the same names and the same parameters/return values. The functionality of the services will however be different because of the simulation of the services.

The data transmission phase of the CO-Socket interface SHALL be mapped down to the PM-DATA.req primitive of the P_Mul sublayer and use the “multicast” service of the ACP 142 protocol (see ref. 3 for details).

When using the Multicast Socket interface, a static list of IP addresses related to each multicast address (or group) has to be present and accessible by the Tactical Adaptaion Sublayer in order to populate the field List_Of_Destination_IDs of the P_Mul PDUs.

4.2.4 Signaling of QoS

For all of the three interfaces defined in the previous section, it SHALL be possible to signal the priority of the data to be transferred. This priority is to be mapped to the Type Of Service (TOS) field in the IP protocol.

Each socket class in JAVA™ have a get/set method for each option it supports, taking and returning the appropriate type.

One of these options is called IP_TOS and is used to set the type-of-service or traffic class field in the IP header for a TCP or UDP socket.

The options are defined in the public interface SocketOptions, and the methods to get and set the values are called getOption and setOption respectively. The methods and constants, which specify options in this interface are for implementation only. If you're not subclassing SocketImpl or DatagramSocketImpl, you won't use these directly**.** There are type-safe methods to get/set each of these options in Socket, *ServerSocket, DatagramSocket* and *MulticastSocket.*

4.2.5 If the receiver is not listening to the port

There may be situations where the receiver is not listening to the port(s) for incoming data. In these cases the recipient SHOULD store the PM-DATA.indication primitives until the Accept() is activated and the application is ready to receive the data. If retransmissions occur, the receiver must make sure not to store the primitives several times.

4.2.6 Error Handling

Reception of PM-P-ABORT.ind or PM-U-ABORT.ind or any other errors or aborts SHOULD be handled gracefully and be mapped to the simulated JAVA™ socket API as related

"exceptions" in order for this API to be as close to the real JAVA™ socket API (defined by java.net package) as possible.

4.2.7 Other related classes, methods, fields and constructors

Other related classes, methods, fields and constructors of the JAVA™ socket API, SHALL be simulated in a way that makes the simulated JAVA™ socket API as close to the real JAVA™ socket API (defined by java.net package) as possible.

4.3 Service mapping between the CO-Socket interface and the P_Mul sublayer

The figures 4.1 and 4.2 show gives an illustration of the service mapping between the CO-Socket interface and the P_Mul sublayer. The connection oriented TCP communication is simulated by the Tactical Adaptation Sublayer. As mentioned before there are several ways for the application to communicate using the socket API in JAVA™, so this should only be regarded as an illustration of the functionality.

4.3.1 Connect()

When the application invokes the *Connect() method* (or a *Socket() method* with automatic connetion establishment), it expects the service to cause a TCP connection establishment using a three-way-handshake. What actually happens is that when the service-user invokes the *Connect() method* (or a related *socket method* involving connection establishment), the Tactical Adaptation Sub-Layer immediately returns with a positive value, and waits for a data transfer phase (i.e. *println()*) containing the real data to be transferred. No TCP SYN segment is sent to the server.

The data is transferred to the Tactical Adaptation Sub-Layer at the peer side using the PM-DATA.req service primitive. The peer Tactical Adaptation Sublayer then causes the "waiting a*ccept() method*" to return with a positive value. No TCP SYN_ACK segment is returned from the server and no ACK segment is sent from the client in response. Thus the connection establishment services are "faked" and the applications do not need to be changed.

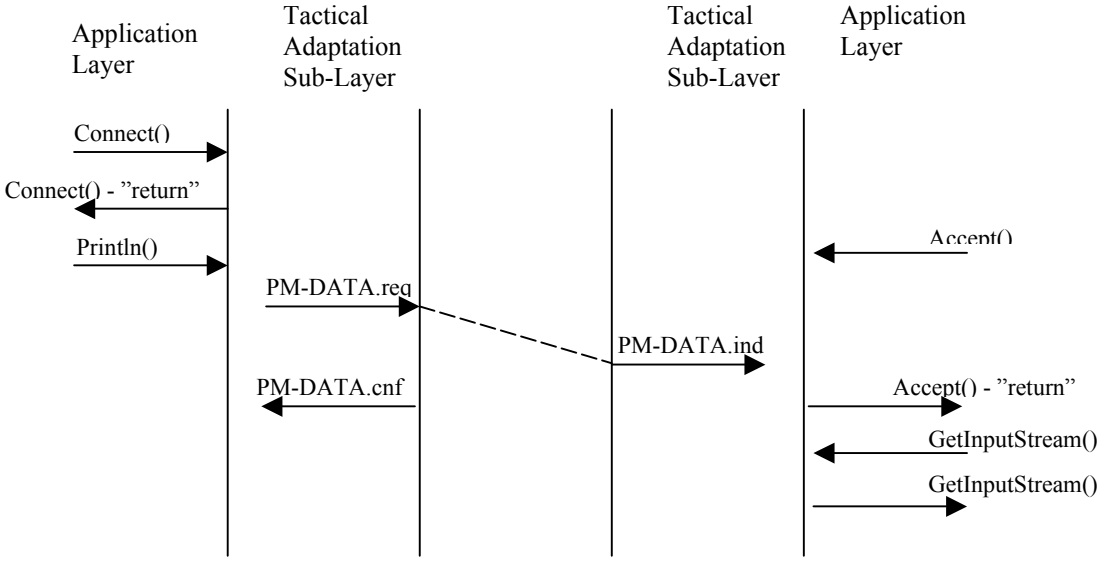

Figure 4.1 Invocation of the Connect() method

4.3.2 Close()

The invocation of the *close() method* is will cause the method to "return" after it has built the Tactical Adaptation Sublayer Data Type and set the field "ProtocolSignalingInformation" to "1" (which is used to indicate a Close() to the peer side (see section 4.1.5)).

When the TA-Sublayer receives a PM-DATA.indication with the "ProtocolSignalingInformation" field of the "Tactical Adaptation Sublayer Data Type" set to "1", it SHALL cause an exception to the application, identical to the exception caused when a "FIN" segment is received in a "normal" TCP connection. No ACK segment SHALL be returned from the receiving side as a response to the request to "close" the simulated connection.

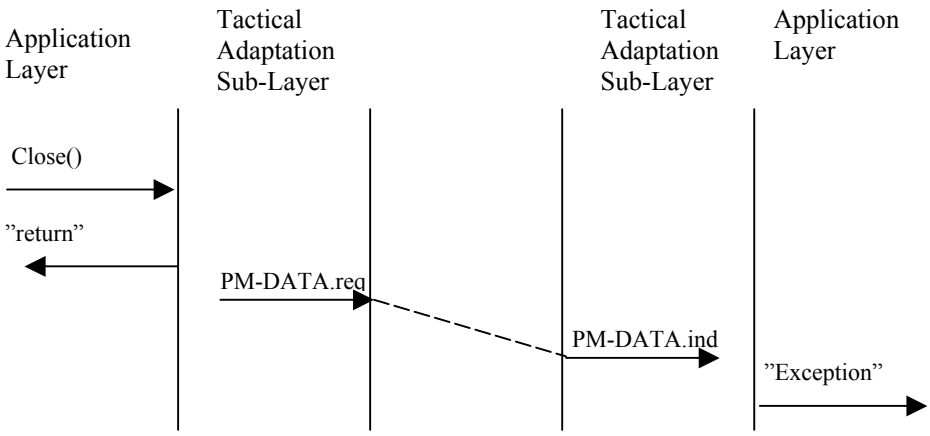

Figure 4.2 Invocation of the Close() method

4.4 Compression

Compression shall be performed at the Tactical Application Sub-layer in order to compress the data. In this section we have defined the compression wrapper as a part of the Tactical Adaptation Sublayer Data Type (TASDT) (see section 4.1.5). This Data Type allows for any OCTET STRING to be compressed, and shall be used to compress the data to be transferred.

4.5 The Tactical Adaptation Sublayer Data Type

The Tactical Adaptation Sublayer Data Type (TASDT) consists of the following fields:

- PortNumber
- ProtocolSignalingInformation
- CompressedData

The fields of the Data Type have the following meaning:

portNumber **(dynamically mandatory)** is used to indicate the port number in order to "route" the data to the right application at the receiving side.

protocolSignalingInformation **(mandatory)** is used to signal any simulated protocol behaviour which is to occur at the receiving side.

CompressedData **(dynamically mandatory)** contains the data to be compressed (see 4.1.5.1).

compressionAlgorithm **(dynamically mandatory)** defines the compression algorithm to be used. The algorithm may be defined using either an INTEGER value (which is **mandatory** to support both on origination and reception) or an OBJECT IDENTIFIER (which is **optional** on origination and **mandatory** on reception).

compressedContentInfo **(dynamically mandatory)** defines the type of content that is compressed. The type of content may be indicated using either an INTEGER value (which is **mandatory** to support both on origination and reception) or an OBJECT IDENTIFIER (which is **optional** on origination and **mandatory** on reception).

compressedContent **(dynamically mandatory)** is the compressed content.

content of any type that is compressed using a specified algorithm. The following object identifier identifies the Compressed Data Type:

```
id-TASDT ID ::= {iso(1) identified-organization(3) nato(26) stanags(0)
                mmhs(4406)object-identifiers(0) id-mcont(4) 3}
```

The Compressed Data Type are defined by the following ASN.1 (ref. (12)) type:

```
DEFINITIONS ::=
BEGIN
IMPORTS –the related information objects

port-number ::= INTEGER;

protocol-Signaling-Information ::= INTEGER  {
      Close (0),
      Connect (1) -- For further use}

CompressedData ::= SEQUENCE {
      compressionAlgorithm  CompressionAlgorithmIdentifier,
      compressedContentInfo CompressedContentInfo }

CompressionAlgorithmIdentifier ::= CHOICE {
      algorithmID-ShortForm  [0] AlgorithmID-ShortForm,
      algorithmID-OID        [1] OBJECT IDENTIFIER }

AlgorithmID-ShortForm ::= INTEGER  {
      zlibCompress (0) }

CompressedContentInfo ::= SEQUENCE {
      CHOICE {
            contentType-ShortForm [0] ContentType-ShortForm,
            contentType-OID       [1] OBJECT IDENTIFIER
      },
      compressedContent [0] EXPLICIT OCTET STRING }
```

```
ContentType-ShortForm ::= INTEGER  {
      unidentified (0),
      external  (1)           -- identified by the object-identifier of
                              -- the EXTERNAL content

--The related information objects
       }

      }

END
```

4.5.1 Use of the Compressed Data Type

To ensure interoperability, this section defines how the information objects SHALL be conveyed within the Compressed Data Type.

The compressed information objects SHALL be placed in the *compressedContent* field of the *CompressedContentInfo* element. The information object SHALL be placed in either the *contentType-ShortForm* or the *contentType-OID* field of the *CompressedContentInfo* element.

An illustration of this required wrapping convention is shown in figure 4.3.

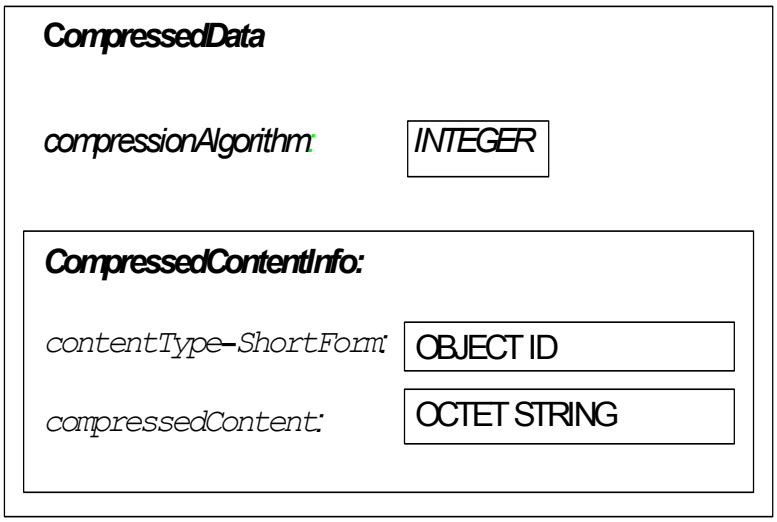

Figure 4.3 Compression protocol wrapping

4.5.2 Compression Algorithm

This specification mandates the support of the compression algorithm ZLIB (ref. (6)) (ref. (7)), which is free of any intellectual property restrictions and has a freely available, portable and efficient reference implementation. The following object identifier identifies ZLIB:

id-alg-zlibCompress OBJECT IDENTIFIER ::= { iso(1) member-body(2) us(840) rsadsi(113549) pkcs(1) pkcs-9(9) smime(16) alg(3) 8 }

The INTEGER reference SHOULD however be used and the integer value 0 identifies this algorithm.

5 THE P_MUL SUB-LAYER

P_Mul (ACP 142) (ref. (3)) is an application layer protocol that is designed to be used together with other protocols to handle EMCON (Emission Control) conditions and multicast communication techniques. It is also a provider of a reliable acknowledged, connectionless application-layer service. It operates directly above a connectionless transport layer and may operate in simplex, half-duplex or full-duplex mode. P_Mul is thus a flexible protocol that may be used by most applications.

Most of the applications are based on the service support from connection oriented TCP protocol for reliable data transport between application entities. This means that these applications cannot be used over simplex connections, or to send the replication data to users under EMCON conditions if they are using TCP. When EMCON conditions apply, some nodes are only allowed to receive data and are not allowed to acknowledge them.

EMCON conditions are handled in P_Mul by allowing acknowledgements from the receiving nodes to be missing for a rather long time. The sending node has to know which of the receiving nodes that are in EMCON, and retransmissions are performed to increase the probability that the nodes in EMCON receive the data. P_Mul must therefore be run on top of a connectionless protocol stack.

By making use of the broadcast properties of a connectionless protocol stack, one data replication may be sent to N recipients instead of replicating the data N times. To handle the problem of packet flooding in broadcast networks, a multicast addressing scheme has been invented.

If there are any conflicts between this Annex and ACP-142, this annex takes precedence.
The complete specification of the P_Mul protocol with procedures, are given in ACP 142 (ref. (3)).

5.1.1 Protocol Data Units

Both NATO and the Combined Communications and Electronics Board (CCEB) has adopted P_Mul and it has been issued as a military standard defined by ACP 142 (ref. (3)).

The ACP 142 specification describes two groups of Protocol Data Units (PDUs). One group consists of those PDUs needed for the transfer of the data. These PDUs are:

- **Address_PDU**
- **Data_PDU**
- **Ack_PDU**
- **Discard_PDU**

The other group consists of PDUs for dynamic configuration of multicast groups. The concept of multicast groups is introduced to reduce the network load in situations when the sender has exact knowledge about the addresses of the recipients. The objective is that the multicast transmission of the replicated data shall involve as few recipients as possible, by forming groups of transmitting and receiving nodes within a multicast network. The Application Protocol Data Units (APDUs) used to manage multicast groups are:

- **Request_PDU** – for requesting a multicast group
- **Reject_PDU –** for rejecting a multicast group
- **Release_PDU** – for releasing a multicast group
- **Announce_PDU** – announcing a multicast group

These four PDUs will be used by a P_Mul management function and not by the data transfer service user. These PDUs SHALL NOT be implemented as a part of this specification. For more details about the P_MUL protocol and the PDUs, see ACP 142 (ref. (3)).

5.2 The P_Mul Sub-Layer Service Interface

In order to make P_Mul more independent of the application protocols using it and to make P_Mul fit into the layered model described in this document, we have defined a service interface with a set of service primitives to be invoked by the P_Mul user. We have defined the following services for the P_Mul Sub-Layer:

Primitives for handling data transmission and errors:
- PM-DATA.request/indication/confirmation(*)
- PM-P-ABORT.indication
- PM-U-ABORT.request/indication

Primitives for handling multicast groups dynamically:
- PM-REQUEST.request/indication
- PM-REJECT.request/indication
- PM-RELEASE.request/indication
- PM-ANNOUNCE.request/indication

(*) This is not a symmetric service, a PM-DATA.confirmation primitive is issued when the PDU is sent to the recipients.

The PM-DATA, PM-P-ABORT and PM-U-ABORT services are used by the Tactical Adaptation Sub-Layer for data transfer and error handling.

The PM-REQUEST, PM-REJECT, PM-RELEASE and PM-ANNOUNCE services are not used by the Tactical Adaptation sub-layer, but by a P_MUL management function directly in order to set up and organise multicast groups dynamically. The reason for defining these four service primitives, is to clearly separate the P_Mul protocol machine from the P_Mul

Management function which may be integrated with the user application. How to handle multicast groups is a local implementation matter and these primitives SHALL NOT be implemented as a part of this specification.

Figure 5.1 shows the layered structure of the protocol profile where we see that some of the P_Mul services are used by the Tactical Adaptation Sublayer, and some services are used by the P_Mul Management Function to handle the multicast groups.

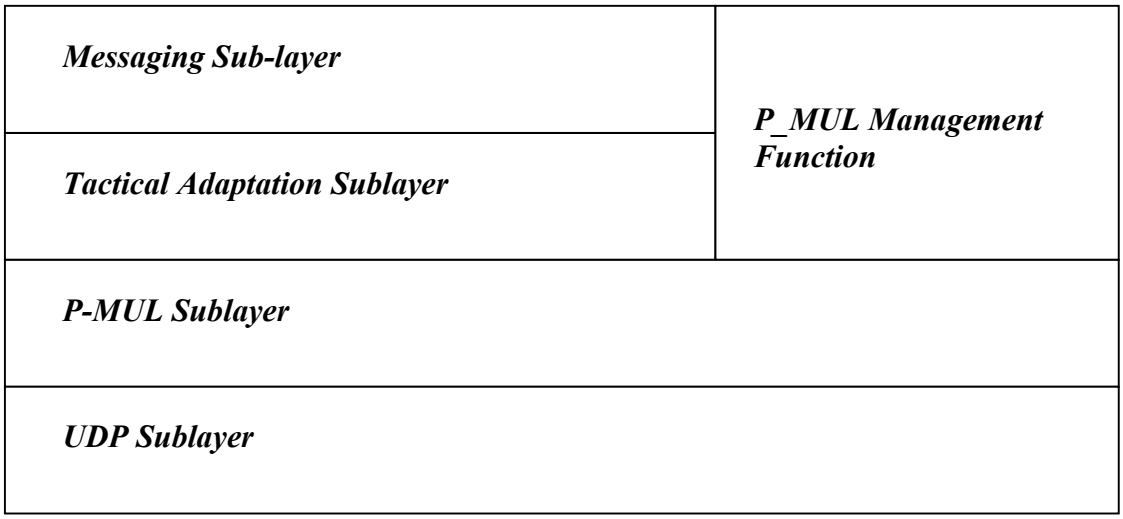

Figure 5.1 The P_Mul Sub-Layer interfaces both the Tactical Adaptation Sub-layer and a P_Mul Management Function

5.3 The P_Mul Sub-Layer Service Primitives and Parameters

5.3.1 PM-DATA

This service is used to send data from the originator to the receiver.
When the P_Mul Sub-Layer receives a PM-DATA.request primitive, it will contain the data to be sent and the sublayer SHALL create and send Address_PDUs and Data_PDUs according to the protocol description in ACP 142.

The PM-DATA.indication primitive is issued by the P_Mul Sublayer to the Tactical Adaptation Sublayer when all of the Data_PDUs belonging to a data replication are received, and an Ack_PDU is sent back indicating no missing Data_PDUs. See ACP-142 for description of the protocol.

A PM-DATA.confirmation primitive is issued by the P_Mul Sub-Layer to the Tactical Adaptation Sublayer when the Data_PDU is sent to the recipients. This is not a symmetric service in that there is no PM-DATA.response primitive. See ACP-142 for description of the protocol. It is important to be aware of that this only acknowledges that the data was sent to the P_Mul Sublayer of the next LDSA and is not to be regarded as an end-to-end acknowledgement between the LDSAs.

Primitive / Parameter	PM-DATA *req*	*ind*	*res*	*cnf*
Priority	M	M(=)	-	M(=)
MessageID	M	M(=)	-	M(=)
Expiry_Time	O	O(=)	-	-
List_of_Destination_Entries	M	P	-	-

Priority
This parameter is to be mapped to the priority fields of the Address_PDU and the Data_PDU. See ACP-142 for description of the field and the semantic.

MessageID
See ACP-142 for details.

Expiry_Time
This parameter is to be mapped to the Expiry_Time field of the Address_PDU. See ACP-142 for description of the field and the semantic.

List_of_Destination_Entries
This parameter is to be mapped to the List_of_Destination_Entries field of the Address_PDU. See ACP-142 for description of the field and the semantic.

5.3.2 PM-P-ABORT

The PM-P-ABORT.indication primitive is issued by the P_Mul Sublayer to the Tactical Adaptation Sub-Layer if an error occurs in the sub-layer and the processing of the data has to be aborted. This primitive is also issued when the T-Derror.indication primitive is received from the WAP WDP layer.

Primitive / Parameter	PM-P-ABORT *req*	*ind*
Reason_Code	-	M

Reason_Code
The Reason_Code is a parameter indicating the reason for the abortion of the data processing caused by the P_Mul Sub-Layer. The Reason_Code may have the following values:

1 - Error receiving the data
2 - Error sending a the data
3 – Unknown error

5.3.3 PM-U-ABORT

The reception of a PM-U-ABORT.request indicates that an error has occurred in the above sub-layers, which has caused the data processing to be aborted. The P_Mul Sub-Layer shall create and send a Discard_Message_PDU according to the protocol description in ACP 142.

The PM-U-ABORT.indication primitive, is issued by the P_Mul Sub_Layer to the Tactical Adaptation Sub-Layer when a Discard_Message_PDU is received. See ACP-142 for description this PDU.

Primitive **Parameter**	**PM-U-ABORT** ***req***	 ***ind***
Priority	M	M(=)
MessageID	M	M(=)

Priority
This parameter is to be mapped to the priority field of the Discard_Message_PDU. See ACP-142 for description of the field and the semantic.

MessageID
See ACP-142 for details.

5.3.4 Use of The UDP Services

The UDP protocol SHALL be used for transmission of all of the PDUs created by the P_Mul Sublayer. The UDP port numbers to be used are defined in ACP-142 Annex B.

The errors of the socket interface used SHALL be mapped onto the PM-P-ABORT.indication primitive. The Error Code parameter is of local significance only and SHOULD be mapped to the Reason Code of the PM-P-ABORT.indication primitive.

6 THE UDP TRANSPORT LAYER

The connection-less transport protocol UDP IETF RFC 768 (ref. (5)) SHALL be used for transmission of all of the PDUs created by the P_Mul Sublayer. The UDP port numbers to be used are defined in ACP-142 Annex B.

UDP provides port based addressing and IP provides the segmentation and reassembly in a connectionless datagram service. UDP also provides a checksum over both header and data.

UDP does not provide any guarantee of delivery and the reliability of packet delivery is therefore left to the P_Mul protocol defined in ACP 142. In all cases where the IP protocol is available over a bearer service the UDP SHALL be used. UDP is fully specified in IETF RFC 768 (ref. (5)), while the IP networking layer is defined in the IETF RFC 791(ref. (4)).

The errors of the socket interface used, SHALL be mapped onto the PM-P-ABORT.indication primitive. The Error Code parameter is of local significance only and SHOULD be mapped to the Reason Code of the PM-P-ABORT.indication primitive.

7 THE IP NETWORK PROTOCOL

The Internet Protocol (IP) IETF RFC 791 (ref. (4)) SHALL be used as the network protocol. It SHALL be possible to use the TOS field for setting and utilizing priority for packet transfer. The vendor SHALL propose a solution for the use of TOS field, based on mechanisms defined by the IETF DiffServ standards.

8 REFERENCES

(1) AC/322(SC/5)N/224, Ratification Draft of STANAG 4406 (Ed. 1): Military Message Handling System, Annex E: Tactical MMHS Protocol and Profile Solution, Editor Anders Eggen (FFI, Norway)

(2) (1998): C. Kenneth Miller, CTO: Data Distribution Over IP in High Error Rate Military Environments , IEEE MILCOM 1998.

(3) ACP 142, P_Mul: A protocol for reliable multicast messaging in bandwidth constrained and delayed acknowledgement (EMCON) environments.

(4) IETF RFC 791 Internet Protocol (IP)

(5) IETF RFC 768 User Datagram Protocol (UDP)

(6) IETF RFC 1950 ZLIB Compressed Data Format Specification version 3.3

(7) IETF RFC 1951 Compressed Data Format Specification version 1.3

(8) IETF RFC 2119 “Key words for use in RFCs to Indicate Requirement Levels”

(9) Eggen, A, “A Protocol Solution for Replication of Information in a NATO Tactical Directory”, FFI/RAPPORT-2003/01517.

(10) http://java.sun.com/j2se/1.4.2/docs/api/java/net/package-summary.html

(11) ISO10731 “Information Technology - Open Systems Interconnection - Basic Reference Model - Conventions for the Definition of OSI Services”, ISO/IEC 10731:1994.

(12) (1990): ISO/IEC: IS 8825, Information Technology - Open Systems Interconnection - Specification of Basic Encoding Rules for Abstract Syntax Notation One (ASN.1).