WAP WTP

Version 11-June-1999

Wireless Application Protocol
Wireless Transaction Protocol Specification

Disclaimer:

This document is subject to change without notice.

Contents

1. Scope

A transaction protocol is defined to provide the services necessary for interactive "browsing" (request/response) applications. During a browsing session, the client requests information from a server, which MAY be fixed or mobile, and the server responds with the information. The request/response duo is referred to as a "transaction" in this document. The objective of the protocol is to reliably deliver the transaction while balancing the amount of reliability required for the application with the cost of delivering the reliability.

WTP runs on top a datagram service and optionally a security service. WTP has been defined as a light weight transaction oriented protocol that is suitable for implementation in "thin" clients (mobile stations) and operates efficiently over wireless datagram networks. The benefits of using WTP include:

- Improved reliability over datagram services. WTP relieves the upper layer from re-transmissions and acknowledgements which are necessary if datagram services are used.
- Improved efficiency over connection oriented services. WTP has no explicit connection set up or teardown phases.
- WTP is message oriented and designed for services oriented towards transactions, such as “browsing”.

2. Document Status

This document is available online in the following formats:

- PDF format at http://www.wapforum.org/.

2.1 Copyright Notice

2.2 Errata

Known problems associated with this document are published at http://www.wapforum.org/.

2.3 Comments

Comments regarding this document can be submitted to the WAP Forum in the manner published at http://www.wapforum.org/.

3. References

3.1 Normative References

[FLEX] FLEX™ Protocol Specification and FLEX™ Encoding and Decoding Requirements, Version G1.9, Document Number 68P81139B01, March 16, 1998, Motorola.

[FLEXSuite] FLEX™ Suite of Application Protocols, Version 1.0, Document Number 6881139B10, October 29, 1997, Motorola.

[GSM0260] ETSI European Digital Cellular Telecommunication Systems (phase 2+) : General Packet Radio Service (GPRS) - stage 1 (GSM 02.60)

[GSM0290] ETSI European Digital Cellular Telecommunication Systems (phase 2) : Unstructured Supplementary Service Data(USSD) - stage 1 (GSM 02.90)

[GSM0340] ETSI European Digital Cellular Telecommunication Systems (phase 2+) : Technical realisation of the Short Message Service (SMS) Point-to-Point (P) (GSM 03.40)

[GSM0360] ETSI European Digital Cellular Telecommunication Systems (phase 2+) : General Packet Radio Service (GPRS) - stage 2 (GSM 03.60)

[GSM0390] ETSI European Digital Cellular Telecommunication Systems (phase 2) : Unstructured Supplementary Service Data(USSD) - stage 2 (GSM 03.90)

[GSM0490] ETSI European Digital Cellular Telecommunication Systems (phase 2) : Unstructured Supplementary Service Data(USSD) - stage 3 (GSM 04.90)

[IS130] EIA/TIA IS-130

[IS135] EIA/TIA IS-135

[IS136] EIA/TIA IS-136

[IS176] EIA/TIA IS-176 - CDPD 1.1 specifications

[IS637] TIA/EIA/IS-637: Short Message Services for Wideband Spread Spectrum Cellular Systems

[ISO7498] ISO 7498 OSI Reference Model

[ISO8509] ISO TR 8509 Service conventions

[ReFLEX] ReFLEX25 Protocol Specification Document, Version 2.6, Document Number 68P81139B02-A, March 16, 1998, Motorola.

[RFC2119] S. Bradner "Keywords for use in RFCs to Indicate Requirement Levels", RFC2119, http://www.internic.net/rfc/rfc2119.txt

[TR45.3.6] General UDP Transport Teleservice (GUTS) ñ Stage III, TR45.3.6/97.12.15

[WAE] "Wireless Application Environment Specification", WAP Forum. URL: http://www.wapforum.org/

[WAPARCH] "Wireless Application Protocol Architecture Specification", WAP Forum. URL: http://www.wapforum.org/

[WAPUD] "WAP and GSM USSD", WAP Forum. URL: http://www.wapforum.org/

[WDP] "Wireless Datagram Protocol Specification", WAP Forum. URL: http://www.wapforum.org/

[WSP] "Wireless Session Protocol Specification", WAP Forum. URL: http://www.wapforum.org/

3.2 Informative References

[RFC768] J. Postel "User Datagram Protocol", RFC768, August 1980, http://www.internic.net/rfc/rfc768.txt

[RFC791] J. Postel "IP: Internet Protocol", RFC791, http://www.internic.net/rfc/rfc791.txt

4. Definitions and abbreviations

4.1 Definitions

For the purpose of this document the following definitions apply.

Device Address
The unique network address assigned to a device and following the format defined by an international standard such as E.164 for MSISDN addresses, X.121 for X.25 addresses or RFC 791 for IPv4 addresses. An address uniquely identifies the sending and/or receiving device.

Initiator
The WTP provider initiating a transaction is referred to as the Initiator.

Mobile Device
Refers to a device, such as a phone, pager, or PDA, connected to the wireless network via a wireless link. While the term ‘mobile’ implies the device is frequently moving, it MAY also include fixed or stationary wireless devices (i.e. wireless modems on electric meters) connected to a wireless network.

Network Type
Network type refers to any network, which is classified by a common set of characteristics (i.e. air interface) and standards. Examples of network types include GSM, CDMA, IS-136, iDEN, FLEX, ReFLEX, and Mobitex. Each network type MAY contain multiple underlying bearer services.

Protocol Control Information (PCI)
Information exchanged between WTP entities to coordinate their joint operation.

Protocol Data Unit (PDU)
A unit of data specified in the WTP protocol and consisting of WTP protocol control information and possibly user data.

Responder
The WTP provider responding to a transaction is referred to as the Responder.

Service Data Unit (SDU)
Unit of information from an upper level protocol that defines a service request to a lower layer protocol.

Service Primitive
An abstract, implementation independent interaction between a WTP user and the WTP provider.

Transaction
The transaction is the unit of interaction between the Initiator and the Responder. A transaction begins with an invoke message generated by the Initiator. The Responder becomes involved with a transaction by receiving the invoke. In WTP several transaction classes have been defined. The invoke message identifies the type of transaction requested which defines the action required to complete the transaction.

User Data
The data transferred between two WTP entities on behalf of the upper layer entities (e.g. session layer) for whom the WTP entities are providing services.

WTP Provider
An abstract machine which models the behavior of the totality of the entities providing the WTP service, as viewed by the user.

WTP User
An abstract representation of the totality of those entities in a single system that make use of the WTP service. Examples of WTP users include the WAP session protocol WSP or an application that runs directly onto WTP.

4.2 Abbreviations

For the purposes of this specification the following abbreviations apply.

API	Application Programming Interface
CDMA	Code Division Multiple Access
CDPD	Cellular Digital Packet Data
CSD	Circuit Switched Data
DBMS	Database Management System
DCS	Data Coding Scheme
ETSI	European Telecommunication Standardization Institute
GPRS	General Packet Radio Service
GSM	Global System for Mobile Communication
GTR	Group Trailer, indicates the end of packet group
GUTS	General UDP Transport Service
IE	Information Element
iDEN	Integrated Digital Enhanced Network
IP	Internet Protocol
LSB	Least significant bits
MPL	Maximum Packet Lifetime
MSISDN	Mobile Subscriber ISDN (Telephone number or address of device)
MS	Mobile Station
MSB	Most significant bits
PCI	Protocol Control Information
PCS	Personal Communication Services
PLMN	Public Land Mobile Network
R-Data	Relay Data
RTT	Round-Trip Time
SAR	Segmentation and Re-assembly
SMSC	Short Message Service Center
SMS	Short Message Service
SPT	Server Processing Time
TIA/EIA	Telecommunications Industry Association/Electronic Industry Association
PDU	Protocol Data Unit
SAP	Service Access Point
SDU	Service Data Unit
TTR	Transmission Trailer
UDCP	USSD Dialogue Control Protocol
UDH	User-Data Header (see [GSM 03.40])
UDHL	User-Data Header Length (see [GSM 03.40])
UDL	User-Data Length (see [GSM 03.40])
UDP	Unreliable Datagram Protocol
USSD	Unstructured Supplementary Service Data
WAE	Wireless Application Environment
WAP	Wireless Application Protocol
WSP	Wireless Session Protocol
WTP	Wireless Transaction Protocol
WDP	Wireless Datagram Protocol

4.3 Requirements

This specification uses the following words for defining the significance of each particular requirement:

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", mean that the definition is an absolute prohibition of the specification.

SHOULD

This word, or the adjective "RECOMMENDED", mean that there MAY exist valid reasons in particular circumstances to ignore a particular item, but the full implications MUST be understood and carefully weighed before choosing a different course.

SHOULD NOT

This phrase, or the phrase "NOT RECOMMENDED" mean 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", mean 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 vein 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.)

5. Protocol Overview

5.1 Protocol Features

The following list summarizes the features of WTP.

- Three classes of transaction service:
  - Class 0: Unreliable invoke message with no result message
  - Class 1: Reliable invoke message with no result message
  - Class 2: Reliable invoke message with exactly one reliable result message
- Reliability is achieved through the use of unique transaction identifiers, acknowledgements, duplicate removal and re-transmissions.
- No explicit connection set up or tear down phases. Explicit connection open and/or close imposes excessive overhead on the communication link.
- Optionally user-to-user reliability: the WTP user confirms every received message.
- Optionally, the last acknowledgement of the transaction MAY contain out of band information related to the transaction. For example, performance measurements.
- Concatenation MAY be used, where applicable, to convey multiple Protocol Data Units in one Service Data Unit of the datagram transport.
- Message orientation. The basic unit of interchange is an entire message and not a stream of bytes.
- The protocol provides mechanisms to minimize the number of transactions being replayed as the result of duplicate packets.
- Abort of outstanding transaction, including flushing of unsent data both in client and server. The abort can be triggered by the user canceling a requested service.
- For reliable invoke messages, both success and failure is reported. If an invoke can not be handled by the Responder, an abort message will be returned to the Initiator instead of the result.
- The protocol allows for asynchronous transactions. The Responder sends back the result as the data becomes available.

5.2 Transaction Classes

The following subsections describes the transaction classes of WTP. The WTP provider initiating a transaction is referred to as the Initiator. The WTP provider responding to a transaction is referred to as the Responder. The transaction class is set by the Initiator and indicated in the invoke message sent to the Responder. Transaction classes can not be negotiated.

5.2.1 Class 0: Unreliable invoke message with no result message

Class 0 transactions provide an unreliable datagram service. It can be used by applications that require an "unreliable push" service. This class is intended to augment the transaction service with the capability for an application using WTP to occasionally send a datagram within the same context of an existing session using WTP. It is not intended as a primary means of sending datagrams. Applications requiring a datagram service as their primary means of data delivery SHOULD use WDP [WDP].

The basic behavior for class 0 transactions is as follows: One invoke message is sent from the Initiator to the Responder. The Responder does not acknowledge the invoke message and the Initiator does not perform re-transmissions. At the Initiator, the transaction ends when the invoke message has been sent. At the Responder, the transaction ends when the invoke has been received. The transaction is stateless and can not be aborted.

5.2.2 Class 1: Reliable invoke message with no result message

Class 1 transactions provide a reliable datagram service. It can be used by applications that require a "reliable push" service.

The basic behavior for class 1 transactions is as follows: One invoke message is sent from the Initiator to the Responder. The invoke message is acknowledged by the Responder. The Responder maintains state information for some time after the acknowledgement has been sent to handle possible re-transmissions of the acknowledgement if it gets lost and/or the Initiator re-transmits the invoke message. At the Initiator, the transaction ends when the acknowledgement has been received. The transaction can be aborted at any time.

If the User acknowledgement function is enabled, the WTP user at the Responder confirms the invoke message before the acknowledgement is sent to the Initiator.

5.2.3 Class 2: Reliable invoke message with one reliable result message

Class 2 transactions provide the basic invoke/response transaction service. One WSP session MAY consist of several transactions of this type.

The basic behavior for class 2 transactions is as follows: One invoke message is sent from the Initiator to the Responder. The Responder replies with exactly one result message that implicitly acknowledges the invoke message. If the Responder takes longer to service the invoke than the Responder's acknowledgement timer interval, the Responder MAY reply with a "hold on" acknowledgement before sending the result message. This prevents the Initiator from unnecessarily re-transmitting the invoke message. The Responder sends the result message back to the Initiator. The result message is acknowledged by the Initiator. The Initiator maintains state information for some time after the acknowledgement has been sent. This is done in order to handle possible re-transmissions of the acknowledgement if it gets lost and/or the Responder re-transmits the result message. At the Responder the transaction ends when the acknowledgement has been received. The transaction can at any time be aborted.

If the User acknowledgement function is enabled, the WTP user at the Responder confirms the invoke message before the result is generated. The WTP user at the Initiator confirms the result message before the acknowledgement is sent to the Responder.

5.3 Relation to Other Protocols

This chapter describes how WTP relates to other WAP protocols. For a complete description of the WAP Architecture refer to [WAP]. The following table illustrates the where the services provided to the WTP user are located.

WTP User (e.g. WSP)	
WTP	❑ Transaction handling ❑ Re-transmissions, duplicate removal, acknowledgements ❑ Concatenation and separation
[WTLS]	❑ Optionally compression ❑ Optionally encryption ❑ Optionally authentication
Datagram Transport (e.g. WDP)	❑ Port number addressing ❑ Segmentation and re-assembly (if provided) ❑ Error detection (if provided)
Bearer Network (e.g. IP, GSM SMS/USSD, IS-136 GUTS)	❑ Routing ❑ Device addressing (IP address, MSISDN) ❑ Segmentation and re-assembly (if provided) ❑ Error detection (if provided)

WTP is specified to run over a datagram transport service. The WTP protocol data unit is located in the data portion of the datagram. Since datagrams are unreliable, WTP is required to perform re-transmissions and send acknowledgement in order to provide a reliable service to the WTP user. WTP is also responsible for concatenation (if possible) of multiple protocol data units into one transport service data unit.

The datagram transport for WAP is defined in [WDP]. The datagram transport is required to route an incoming datagram to the correct WDP user. Normally the WDP user is identified by a unique port number. The responsibility of WDP is to provide a datagram service to the WDP user, regardless of the capability of the bearer network type. Fortunately, datagram service is a common transport mechanism, and most bearer networks already provide such a service. For example, for IP-based utilize UDP for this service.

The bearer network is responsible for routing datagrams to the destination device. Addressing is different depending on the type of bearer network (IP addresses or phone numbers). In addition, some networks are using dynamic allocation of addresses, and a server has to be involved to find the current address for a specific device. Network addresses within the WAP stack MAY include the bearer type and the address (e.g. ìIP; 123.456.789.123î). The multiplexing of data to and from multiple bearer networks with different address spaces to the same WAP stack, has not been specified.

5.4 Security Considerations

WTP has no security mechanisms.

5.5 Management Entity

The WTP Management Entity is used as an interface between the WTP layer and the environment of the device. The WTP Management Entity provides information to the WTP layer about changes in the devices environment, which MAY impact the correct operation of WTP.

The WTP protocol is designed around an assumption that the environment in which it is operating is capable of transmitting and receiving data. For example, this assumption includes the following basic capabilities that MUST be

provided by the mobile device:

- the mobile is within a coverage area applicable to the bearer service being invoked;
- the mobile having sufficient power and the power being on;
- sufficient resources (processing and memory) within the mobile are available to WTP;
- the WTP protocol is correctly configured, and ;
- the user is willing to receive/transmit data.

The WTP Management Entity monitors the state of the above services/capabilities of the mobile's environment and would notify the WTP layer if one or more of the assumed services were not available. For example if the mobile roamed out of coverage for a bearer service, the Bearer Management Entity SHOULD report to the WTP Management Entity that transmission/reception over that bearer is no longer possible. In turn, the WTP Management Entity would indicate to the WTP layer to close all active connections over that bearer. Other examples such as low battery power would be handled in a similar way by the WTP Management Entity.

In addition to monitoring the state of the mobile environment the WTP Management Entity MAY be used as the interface to the user for setting various configuration parameters used by WTP, such as device address. It could also be used to implement functions available to the user such as a ìdrop all data connectionsî feature. In general the WTP Management Entity will deal with all issues related to initialization, configuration, dynamic re-configuration, and resources as they pertain to the WTP layer.

Since the WTP Management Entity MUST interact with various components of a mobile device which are manufacturer specific, the design and implementation of the WTP Management Entity is considered outside the scope of the WTP Specification and is an implementation issue.

5.6 Static WTP Conformance Clause

This static conformance clause defines a minimum set of WTP features that can be implemented to ensure that the implementation will be able to interoperate.

The features needed from WTP is dictated by the WTP user. In the case when WSP is the WTP user, it also depends on whether the WSP protocol implementation operates as a client (initiates WSP sessions) or as a server. In the following table Mandatory (M) and Optional (O) features of WTP are listed, in the case when WSP is the user.

Table 1 WTP as a client Static Conformance Requirements

Identifier	WTP Function	Type	Reference	
WTP – SC1	Transaction Class 0	Initiator	7.1.3	M
WTP – SC2		Responder	7.1.3	M
WTP – SC3	Transaction Class 1	Initiator	7.2.4	M
WTP – SC4		Responder	7.2.4	M
WTP – SC5	Transaction Class 2	Initiator	7.3.4	M
WTP – SC6		Responder	7.3.4	O
WTP – SC7	User acknowledgement		8.3	M
WTP – SC8	Concatenation		5.1 8.5	O
WTP – SC9	Separation		8.5	M
WTP – SC10	Re-transmission until acknowledgement		8.2	M
WTP – SC11	Transaction abort		5.6 8.7 8.12	M
WTP – SC12	Error handling		8.12	M

WTP – SC13	Information in last acknowledgement		8.4 8.10	O
WTP – SC14	Asynchronous transactions		8.6	O
WTP – SC15	Transaction identifier verification	Initiator	8.1.5.2	M (Note 1)
WTP – SC16		Responder	8.1.5.2 8.8.1	O (Note 1)
WTP – SC17	Transport Informational Items	Error	8.10	O
WTP – SC18		Info	8.10	O
WTP – SC19		Option	8.10	O
WTP – SC20		PSN	8.10	O
WTP – SC21	Segmentation and Re-assembly with selective re-transmission and packet Groups		8.14	O

Table 2 WTP as a server Static Conformance Requirements

Identifier	**WTP Function**	**Type**	**Reference**	
WTP – SC22	Transaction Class 0	Initiator	7.1.3	M
WTP – SC23		Responder	7.1.3	M
WTP – SC24	Transaction Class 1	Initiator	7.2.4	M
WTP – SC25		Responder	7.2.4	M
WTP – SC26	Transaction Class 2	Initiator	7.3.4	O
WTP – SC27		Responder	7.3.4	M
WTP – SC28	User acknowledgement		8.3	M
WTP – SC29	Concatenation		5.1 8.5	O
WTP – SC30	Separation		8,5	M
WTP – SC31	Re-transmission until acknowledgement		8.2	M
WTP – SC32	Transaction abort		5.6 8.7 8.12	M
WTP – SC33	Error handling		8.12	M
WTP – SC34	Information in last acknowledgement		8.4 8.10	O
WTP – SC35	Asynchronous transactions		8.6	O
WTP – SC36	Transaction identifier verification	Initiator	8.1.5.2	M
WTP – SC37		Responder	8.1.5.2 8.8.1	O
WTP – SC38	Transport Informational Items	Error	8.10	O
WTP – SC39		Info	8.10	O
WTP – SC40		Option	8.10	O
WTP – SC41		PSN	8.10	O
WTP – SC42	Segmentation and Re-assembly with selective re-transmission and packet Groups		8.14	O

Note 1) It is mandatory for the initiator to respond to a verification. For the responder it is optional to initiate a verification.

If the WTP provider is requested to execute a procedure it does not support, the transaction MUST be aborted with the an appropriate error code. For example, a Responder not supporting class 2 receiving a class 2 transaction aborts the transaction with the NOTIMPLEMENTEDCL2 abort code.

Segmentation and re-assembly (SAR) and selective re-transmission MAY be implemented in order to enhance the WTP service. If SAR is not implemented in WTP, this functionality should be provided by another layer in the stack. For example, in IS-136 the SSAR layer handles SAR, in an IP network IP [RFC791] handles SAR and for GSM SMS/USSD SAR is achieved by using SMS concatenation [GSM0340]. The motivation for implementing WTP SAR is the selective re-transmission procedure, which MAY, if large messages are sent, improve the over-the-air efficiency of the protocol.

Whether WTP SAR is supported or not is indicated by the Initiator when the transaction is invoked. The following table shows how WTP Initiators and Responders SHOULD guarantee interoperability between WTP providers that have and those that have not implemented WTP SAR.

Table 3 Interoperability between WTP Providers with and without WTP SAR

	Initiator	
Responder	**WTP SAR**	**Not WTP SAR**
WTP SAR	Full interoperability	Responder MUST NOT respond with a WTP segmented message
Not WTP SAR	Responder abort transaction with the abort code NOTIMPLEMENTEDSAR Initiator MUST re-send the transaction without using WTP SAR	Full interoperability

Note 1) If a Responder not supporting WTP SAR receives a non-segmented message from an Initiator that supports WTP SAR, there is no need to abort the transaction. The Initiator will never be aware of the fact that the Responder does not support WTP SAR.

5.7 Other WTP Users

The intended use of this protocol is to provide WSP [WSP] with a reliable transaction service over an unreliable datagram service. However, the protocol can be used by other applications with similar communication needs.

6. Elements for Layer-to-Layer Communication

6.1 Notations Used

6.1.1 Definition of Service Primitives and Parameters

Communications between layers and between entities within the layer are accomplished by means of service primitives. Service primitives represent, in an abstract way, the logical exchange of information and control between the transaction layer and adjacent layers. They do not specify or constrain implementations.

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 - Generic name . Type (Parameters)

where X designates the layer providing the service. For this specification X is:

"TR" for the Transaction Layer.

An example of a service primitive for the WTP layer would be TR-Invoke Request.

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 are an implementation issue and are beyond the scope of this specification.

6.1.2 Primitives 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 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

6.1.3 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	TR-primitive			
	req	***ind***	***res***	***cnf***
Parameter 1	M	M(=)	-	-

Parameter 2	-	-	O	C(=)

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

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:

Table 4. Parameter Usage Legend

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 ñ it MAY be omitted
P	Presence of the parameter is a service provider option ñ an implementation MAY not provide it
ñ	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

6.2 Requirements on the Underlying Layer

The WTP protocol is specified to run on top of a datagram service. The datagram service MUST handle the following functions:

- Port numbers to route the incoming datagram to the WTP layer;
- Length information for the SDU passed up to the WTP layer.

The datagram service MAY handle the following functions

- Error detection. For example, by using a checksum.

In addition, Segmentation And Re-assembly (SAR) is expected to be provided by the underlying layers. However, it is usually done at a layer below the datagram layer. For example, in an IP network, the IP protocol handles SAR.

6.3 Services Provided To Upper Layer

6.3.1 TR-Invoke

This primitive is used to initiate a new transaction.

Primitive / Parameter	TR-Invoke			
	req	***ind***	***res***	***cnf***
Source Address	M	M (=)		
Source Port	M	M (=)		
Destination Address	M	M (=)		
Destination Port	M	M (=)		
Ack-Type	M	M (=)		
User Data	O	C (=)		
Class Type	M	M (=)		
Exit Info			O	C (=)
Handle	M	M	M	M

6.3.1.1 Source Address

The source address is the unique address of the device making a request to the WTP layer. The source address MAY be an MSISDN number, IP address, X.25 address or other identifier.

6.3.1.2 Source Port

The source port number associated with the source address.

6.3.1.3 Destination Address

The destination address of the user data submitted to the WTP layer. The destination address MAY be an MSISDN number, IP address, X.25 address or other identifier.

6.3.1.4 Destination Port

The destination port number associated with the destination address for the requested or existing transaction.

6.3.1.5 Ack-Type

This parameter is used to turn the User acknowledgement function on or off.

6.3.1.6 User Data

The user data carried by the WTP protocol. The unit of data submitted to or received from the WTP layer is also referred to as the Service Data Unit. This is the complete unit (message) of data which the higher layer has submitted to the WTP layer for transmission. The WTP layer will transmit the Service Data Unit and deliver it to its destination without any manipulation of its content.

6.3.1.7 Class Type

Indicates the WTP transaction class.

6.3.1.8 Exit Info

Additional user data to be sent to the originator on transaction completion.

6.3.1.9 Handle

The transaction handle is an index returned to the higher layer so the higher layer can identify the transaction and associate the data received with an active transaction. The TR-Handle uniquely identifies a transaction. TR-Handle is an alias for the source address, source port, destination address, and destination port of the transaction. The TR-Handle has local significance only.

6.3.2 TR-Result

This primitive is used to send back a result of a previously initiated transaction.

Primitive / Parameter	TR-Result			
	req	***ind***	***res***	***cnf***
User Data	O	C (=)		
Exit Info			O	C (=)
Handle	M	M	M	M

6.3.3 TR-Abort

This primitive is used to abort an existing transaction

Primitive / Parameter	TR-Abort	
	req	***ind***
Abort Code	O	C (=)
Handle	M	M

6.3.3.1 Abort Code

The abort code indicates the reason for the transaction being aborted. This can include abort codes generated by the WTP protocol and user defined local abort codes.

7. Classes of Operation

7.1 Class 0 Transaction

7.1.1 Motivation

Class 0 is an unreliable datagram service. It can be used by WSP [WSP], for example, to make an unreliable “push” within a session using the same socket association.

This class is intended to augment the transaction service with the capability for an application using WTP to occasionally send a datagram within the same context of an existing session using WTP. It is not intended as a primary means of sending datagrams. Applications requiring a datagram service SHOULD use WDP as defined in [WDP].

7.1.2 Protocol Data Units

The following PDU is used:

1. Invoke PDU

7.1.3 Procedure

A Class 0 transaction is initiated by the WTP user by issuing the TR-Invoke request primitive with the Transaction Class parameter set to Class 0. The WTP provider sends the invoke message and becomes the Initiator of the transaction. The remote WTP provider receives the invoke message and becomes the Responder of the transaction. The Initiator does not wait for or expect a response. If the invoke message is received by the Responder it is accepted immediately. There is no duplicate removal or verification procedure performed. However, the initiator MUST increment the TID counter between each transaction, but the responder MUST NOT update it's cached TID.

This transaction class MUST be supported by the WTP provider. The WTP provider MUST be able to act as both Initiator and Responder.

An example of this class can be found in chapter 11.2.

7.2 Class 1 Transaction

7.2.1 Motivation

The Class 1 transaction is a reliable invoke message without any result message. This type of transaction can be used by WSP [WSP] to realize a reliable "push" service.

7.2.2 Service Primitive Sequences

The following table describes legal service primitive sequences. A primitive listed in the column header MAY only be followed by primitives listed in the row headers that are marked with an "X".

Table 5 Primitive Sequence Table for Transaction Class 1

	TR-Invoke				**TR-Abort**	
	req	***ind***	***res***	***cnf***	***req***	***ind***
TR-Invoke.req						
TR-Invoke.ind						
TR-Invoke.res		X				
TR-Invoke.cnf	X					
TR-Abort.req	X	X	X			
TR-Abort.ind	X	X	X			

7.2.3 Protocol Data Units

The following PDUs are used:

1. Invoke PDU
2. Ack PDU
3. Abort PDU

7.2.4 Procedure

A Class 1 transaction is initiated by the WTP user by issuing the TR-Invoke request primitive with the Transaction Class parameter set to Class 1. The WTP provider sends the invoke message and becomes the Initiator of the transaction. The remote WTP provider receives the invoke message and becomes the Responder of the transaction. The Responder checks the Transaction Identifier and determines whether a verification has to be initiated. If not, it delivers the message to the user and returns the last acknowledgement to the Initiator. The Responder MUST keep state information in order to re-transmit the last acknowledgement if it gets lost.

This transaction class MUST be supported by the WTP provider. The WTP provider MUST be able to act as both Initiator and Responder.

An example of this class can be found in chapter 11.3.

7.3 Class 2 Transaction

7.3.1 Motivation

The Class 2 transaction is the basic request/response transaction service. This is the most commonly used transaction service. For example, it is used by WSP [WSP] for method invocations.

7.3.2 Service Primitive Sequences

The following table describes legal service primitive sequences. A primitive listed in the column header MAY only be followed by primitives listed in the row header that are marked with an "X".

Table 6 Primitive Sequence Table for Transaction Class 2

	TR-Invoke				**TR-Result**				**TR-Abort**	
	req	***ind***	***res***	***cnf***	***req***	***ind***	***res***	***cnf***	***req***	***ind***
TR-Invoke.req										
TR-Invoke.ind										
TR-Invoke.res		X								
TR-Invoke.cnf	X									

	TR-Invoke				**TR-Result**				**TR-Abort**	
	req	***ind***	***res***	***cnf***	***req***	***ind***	***res***	***cnf***	***req***	***ind***
TR-Result.req		X*	X							
TR-Result.ind	X*			X						
TR-Result.res						X				
TR-Result.cnf					X					
TR-Abort.req	X	X	X	X	X	X	X			
TR-Abort.ind	X	X	X	X	X	X	X			

* = NOT valid if User acknowledgement is used (see separate section).

7.3.3 Protocol Data Units

The following PDUs are used:

1. Invoke PDU
2. Result PDU
3. Ack PDU
4. Abort PDU

7.3.4 Procedure

A Class 2 transaction is initiated by the WTP user by issuing the TR-Invoke request primitive with the Transaction Class parameter set to Class 2. The WTP provider sends the invoke message and becomes the Initiator of the transaction. The remote WTP provider receives the invoke message and becomes the Responder of the transaction. The Responder checks the Transaction Identifier and determines whether a verification has to be initiated. If not, it delivers the message to the WTP user and wait for the result. The Responder MAY send a hold on acknowledgement after a specified time.

The WTP user sends the result message by issuing the TR-Result request primitive. When the Initiator has received the result message it returns the last acknowledgement to the Responder. The Initiator MUST keep state information in order to re-transmit the last acknowledgement if it gets lost.

If the Responder does not support this transaction class it returns an Abort PDU with the abort reason NOTIMPLEMENTEDCL2 as a response to the invoke message.

An example of this class can be found in chapter 11.4.

8. Protocol Features

8.1 Message Transfer

8.1.1 Description

WTP consists of two types of messages: data messages and control messages. Data messages carry user data. Control messages are used for acknowledgements, error reporting, etc. and do not carry user data. This section gives the reader an overall picture of how transactions are realized by WTP. The procedures to guarantee reliable message transfer are outlined. Special functions like concatenation and separation, re-transmission until acknowledgement, transaction abort, user acknowledgement, and others are described in further detail in separate sections.

It is important to note that not all messages and functions are used by all transaction classes. The following table illustrates which messages are used for the different transaction classes.

Table 7 Summary of WTP message transfer

Message/function	Class 2	Class 1	Class 0
Invoke message	X	X	X (Note 2)
Verification	X	X	
Hold on acknowledgement	X (Note 1)		
Result message	X		
Last acknowledgement	X	X	

Note 1) Only sent in the case when the user takes longer time to service the invoke message than the Responder's acknowledgement timer interval.

Note 2) The class 0 transaction is unreliable. No response is expected from the Responder and no verification is performed.

8.1.2 Service Primitives

The following service primitives are used during nominal WTP transactions. Their use is transaction class dependent:

1. TR-Invoke
2. TR-Result

8.1.3 Transport Protocol Data Units

The following PDUs are used during nominal WTP transactions. It is important to note that not all PDUs are used in every transaction class.

1. Invoke PDU
2. Result PDU
3. Ack PDU

8.1.4 Timer Intervals and Counters

The following timer intervals and counters are used during a nominal WTP transaction. Their use is transaction class dependent.

1. Re-transmission interval
2. Re-transmission counter
3. Acknowledgment interval

4. Wait timeout interval

The values and relations between timer intervals and counters MAY depend on the transaction class being used. A detailed description of timers and counters is provided in a separate section.

8.1.5 Procedure

A transaction takes place between two WTP providers. A WTP user initiates a transaction by issuing the TR-Invoke request primitive. The TCL parameter of the primitive indicates the transaction class: 0, 1 or 2. In WTP, the Initiator is the WTP provider initiating the transaction and the Responder is the WTP provider responding to the initiated transaction.

8.1.5.1 Invoke message

The invoke message is always the first message of a transaction and it is sent using the Invoke PDU. The Initiator administers the Transaction Identifier (TID) by incrementing the TID by one for every initiated transaction. The TID is conveyed in every PDU belonging to the transaction. When the Invoke PDU has been sent the Initiator starts the re-transmission timer and waits for a response. When the Responder receives the Invoke PDU with a valid TID, it delivers the message to the user by generating the TR-Invoke indication primitive.

8.1.5.2 Verification

When the Responder has received and accepted the invoke message it SHOULD cache the TID. This is done in order to filter out duplicate and old invoke messages that have lower or identical TID values (see section on Transaction Identifier). If the Responder determines the TID in the Invoke PDU is invalid, the Responder can verify whether the invoke message is a new or delayed message. This is accomplished by sending an Ack PDU which initiates a three way handshake towards the Initiator (see section on TID Verification). In this case, the Responder MUST NOT deliver the data to the user until the three way handshake is successfully completed. If the three way handshake attempt fails, the transaction is aborted by the Initiator.

8.1.5.3 Hold on acknowledgement

When the invoke message has been delivered to the WTP user, the acknowledgement timer is started. If the WTP user requires more time to service the invoke message than the acknowledgement timer interval, the Responder MAY or SHOULD or MUST send a ìhold onî acknowledgement. This is done to prevent the Initiator from re-transmitting the Invoke PDU. When the Initiator receives the Ack PDU it stops re-transmitting the Invoke PDU and generates the TR-Invoke confirm primitive.

8.1.5.4 Result message

Upon assembling the data, the WTP user sends a result message by initiating the TR-Result request primitive. The result message is transmitted using the Result PDU. When the Result PDU has been sent the Responder starts the re-transmission timer and waits for a response. After the Result PDU is received by the Initiator it generates the TR-Invoke confirm primitive if one has not already been issued and the forwards up the TR-Result indication primitive.

8.1.5.5 Last acknowledgement

The last Ack PDU is sent when the last message of the transaction has been received. The sender of the acknowledgment MUST maintain state information required to handle a re-transmission of the previous message. This can be done by using a wait timer, or by keeping a transaction history that indicates the results of past transactions.

8.2 Re-transmission Until Acknowledgement

8.2.1 Motivation

The re-transmission until acknowledgement procedure is used to guarantee reliable transfer of data from one WTP provider to another in the event of packet loss. To minimize the number of packets sent over-the-air, WTP uses implicit acknowledgements wherever possible. An example of this is the use of the Result message to implicitly acknowledge the Invoke message.

8.2.2 Transport Protocol Data Units

The following PDUs are used:

1. Invoke PDU
2. Result PDU
3. Ack PDU

8.2.3 Timer Intervals and Counters

The following timer intervals and counters are used:

1. Re-transmission interval
2. Re-transmission counter

The values and relationships between timers and counters MAY depend on the transaction class being used. A detailed description of timers and counters is provided in a separate section.

8.2.4 Procedure

When a packet has been sent, the re-transmission timer is started and the re-transmission counter is set to zero. If a response has not been received when the re-transmission timer expires, the re-transmission counter is incremented by one, the packet re-transmitted, and the re-transmission timer re-started. The WTP provider continues to re-transmit until the number of re-transmissions has exceeded the maximum re-transmission value. If no acknowledgement has been received when the retransmission counter is fully incremented and the timer expires, the transaction is terminated and the local WTP user is informed.

The first time a PDU is transmitted the re-transmission indicator (RID) field in the header is clear. For all re-transmissions the RID field is set. Other than the RID field, the WTP provider MUST NOT change any fields in the PDU header.

The motivation for the re-transmission indicator is for the receiver to detect messages that have been duplicated by the network. A WTP provider that receives two identical messages with the RID set to zero, can safely ignore the second message because it must have been duplicated by the network. Any subsequent retransmissions that have the RID flag set to one can not be ignored by the receiver. Re-transmitted messages that gets duplicated by the network must be treated as valid messages by the provider. The receiver in this situation can no longer distinquish between provider retransmissions and network duplicated packets. In this case, if the message is an Invoke PDU, there is a risk that the transaction will be re-played. To avoid such an error, the WTP provider should make a TID validation (see chapter 8.8).

8.3 User Acknowledgement

8.3.1 Motivation

The User Acknowledgement function allows for the WTP user to confirm every message received by the WTP provider.

When this function is enabled, the WTP provider does not respond to a received message until after the WTP user has confirmed the indication service primitive (by issuing the response primitive). If the WTP user does not confirm the indication primitive after a specified time, the transaction is aborted by the provider. Note that this is a much stronger form of a confirmed service than the traditional definition [ISO8509]. The traditional definition of a confirmed service is that there is a confirmation from the service provider, however, there is not necessarily any relationship to a response from the peer service user. In WTP, when the User Acknowledgement function is used, the service provider requires a response from the service user for each indication. As a result, when the confirmation primitive is generated, there is a guarantee that there was a response from the peer service user.

This function is optional within WTP however WSP does utilize the User Acknowledgement feature and therefore any implementation of WTP that will have WSP as the higher layer, must implement it (see 5.6). WSP requires a feature that at the end of a request-response transaction, the server gets a positive indication that the client has processed the response. This is illustrated below.

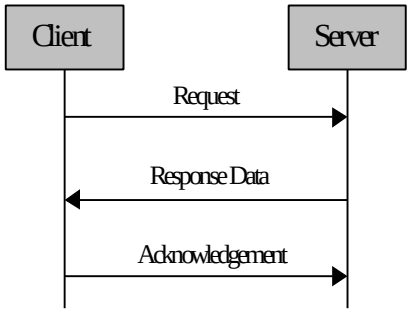


Figure 1 Generic WSP [WSP] transaction

In this model, the *Acknowledgment* is used to convey the fact that the response was received and processed by the client application. It is important to note that the *Client* and the *Server* in the figure refers to the client and server **Application**, and not only the protocol stack.

When the User Acknowledgement function is used the WSP -WTP primitive sequence for a Class 2 transaction becomes as illustrated below.

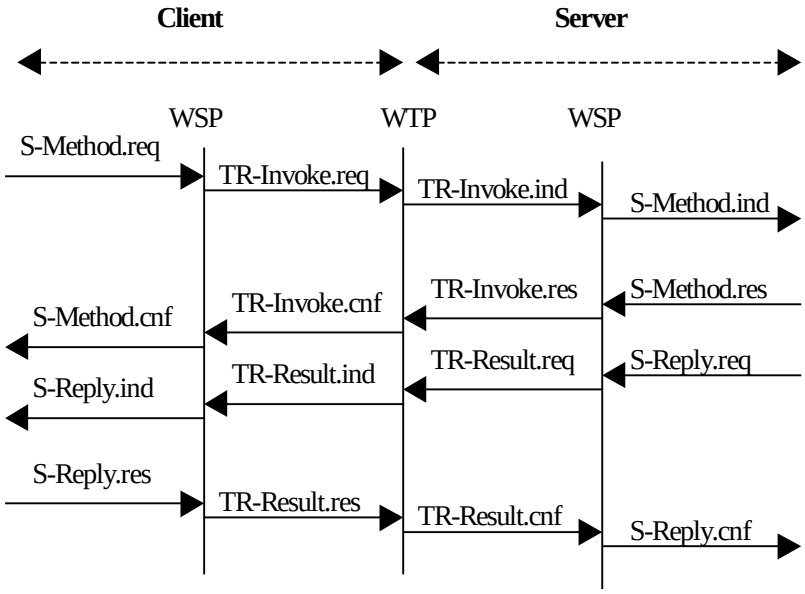


Figure 2 WSP-WTP primitive sequence for request-response

The primitive sequence started by the *S-Reply.res* and *TR-Result.res* primitives realizes the *Complete* and *Confirm* primitives from Figure 1. If the application and/or the WSP for some reason does not issue these primitives, WTP aborts the transaction with the NORESPONSE reason. The abort is used by the WSP server as an indication that the result was not properly received or processed by the client.

The primitive sequence started by the *S-Method.res* and *TR-Invoke.res* primitives can be used by the client WSP to indicate to the application (and human user) that the invoke message has been received by the server WSP.

When this function is not used, WTP MAY acknowledge received messages independently of the WTP user. In Figure 2 this means that the response primitives MAY be ignored by the WTP provider. Put in other words: the WTP provider receives a message, returns an acknowledgement and indicates to the user that a message has been received. If there is an error, the transaction will be aborted by the WTP provider. If the WTP user is alive but can not process the message it MAY abort the transaction with an appropriate abort reason.

This function is optional. It applies to transaction class 1 and 2.

Note) Even though the WTP user has issued a response primitive there is no guarantee that it has interpreted the data and started processing. The WTP user MAY have only copied the data from one buffer to another, or issued the response primitive without any action taken at all. A WTP user can always abort a transaction if it discovers that the received data is corrupt or for some other reason not possible to process (see section on Transaction abort).

8.3.2 Protocol Data Units

The following PDUs are used:

1. Invoke PDU
2. Abort PDU

8.3.3 Procedure

The Initiator sets the U/P-flag in the Invoke PDU to indicate that User acknowledgement is required. A Responder not supporting this function aborts the transaction with the abort reason NOTIMPLEMENTEDUACK. The Initiator MAY then take the decision to re-initiate the transaction without the User acknowledgement function.

When the Responder receives the Invoke PDU with the U/P-flag set it generates the TR-Invoke indication and starts the acknowledgement timer. To give the WTP user time to read the parameters in the indication primitive and issue the TR-Invoke response primitive, the value of the timer MAY have a higher value than the provider's acknowledgement timer (see definitions of default timer values). The Responder MUST NOT return a response before the WTP user has issued the TR-Invoke response primitive. If the Initiator re-transmits Invoke PDUs due to lack of acknowledgement, the Responder MUST silently discard the PDU and restart the acknowledgement timer. When the WTP user issues the TR-Invoke response primitive, the Responder is enabled to send the Ack PDU. If the TR-Invoke response primitive has not been issued after a specified time, the provider aborts the transaction with the abort reason NORESPONSE. If the WTP user issues the TR-Result request primitive, the result is sent instead of the acknowledgement. The Initiator receiving the Ack PDU generates the confirm primitive which indicates that the remote WTP user has issued the corresponding response primitive.

For class 2 transactions, if the Initiator has indicated that the User Acknowledgement function shall be used, it is valid for the entire transaction. This means that when the Initiator has received the result and generated the TR-Result indication primitive it MUST wait for the TR-Result response primitive from the WTP user before the last acknowledgement can be sent. If the TR-Result response primitive has not been issued after a specified time, the provider aborts the transaction with the abort reason NORESPONSE. When the Responder receives the NORESPONSE abort it generates the TR-Abort indication primitive, indicating to the WTP user that the transaction failed.

8.4 Information In Last Acknowledgement

8.4.1 Motivation

The WTP user is allowed to attach information in the last, and only the last, acknowledgement of a transaction. This function is meant for transporting small amounts of information related to the transaction. The information can be, for example, performance measurements collected in order to evaluate the userís perceived quality of service.

For class 2 transactions, this function can be used by the Initiator to communicate some information back to the Responder. For a class 1 transaction, this function can be used by the Responder to communicate some information to the Initiator.

8.4.2 Service Primitives

The following service primitives and parameters are used:

1. TR-Result.res (Class 2)
2. TR-Invoke.res (Class 1)

8.4.3 Protocol Data Units

The following PDU is used:

1. Ack PDU

8.4.4 Procedure

For a class 2 transaction, information is attached to the last acknowledgement by issuing the TR-Result response primitive with the ExitInfo parameter.

For a class 1 transaction, information is attached to the last acknowledgement by issuing the TR-Invoke response primitive with the ExitInfo parameter.

The exit information is transferred as a Transport Information Item (TPI) in the variable part of the Ack PDU header.

For class 2 transactions, the ExitInfo parameter MUST NOT be included in the TR-Invoke response primitive and the Info TPI MUST NOT be included in the Ack PDU that acknowledges the Invoke PDU.

8.5 Concatenation and Separation

8.5.1 Motivation

Concatenation is the procedure to convey multiple WTP Protocol Data Units (PDUs) in one Datagram Service Data Unit (SDU) of the bearer network. When concatenation is done, a special mapping of the WTP PDUs to the SDUs is used. This is described in chapter 9.

Separation is the procedure to extract multiple PDUs from one SDU. When the PDUs have been separated they are dispatched to the transactions.

Concatenation and separation is used to provide over-the-air efficiency, since fewer transmissions over the air are required.

8.5.2 Procedure

Concatenation can only be done for messages with the same address information (source and destination port, source and destination device address).

Concatenation of PDU from different transactions can be done at any time. For example, the last acknowledgement of one transaction can be concatenated with the invoke message of the next transaction. Concatenation and separation is performed outside the WTP state machine.

The exact implementation of concatenation is not specified. Only the structure to be used when multiple packets are concatenated is specified. Exactly how the packets are buffered and concatenated is an implementation issue.

8.6 Asynchronous Transactions

8.6.1 Motivation

The implementation of the WTP provider SHOULD be able to initiate multiple transactions before it receives the response to the first transaction. Multiple transactions SHOULD be handled asynchronously. For example, the responses to transaction number 1, 2 and 3 MAY arrive to the Initiator as 3, 1 and 2. The Responder SHOULD send back the result as soon as it is ready, independently of other transactions.

The maximum number of outstanding transactions at any moment is limited by the maximum number of Transaction Identifiers. The Transaction Identifier is 16 bits, but the high order bit is used to indicate the direction of the message, so

the maximum number of outstanding transactions is 2**15. The implementation environment will also set a limit to how many outstanding transactions it can handle simultaneously.

If the maximum number of outstanding transactions is exceeded the responder should ignore and discard the invoke message.

8.7 Transaction Abort

8.7.1 Motivation

An outstanding transaction can be aborted by the WTP user by issuing the TR-Abort request primitive. The user abort can be triggered by the application (e.g. input from human user) or it can be a negative result (e.g. the WTP user could not generate a result due to an error).

An outstanding transaction can also be aborted by the WTP provider due to a protocol error (e.g. reject the received data) or if a requested function is not implemented.

This function MUST be used with care. If the invoke message has already been sent, the response message MAY be on its way to the client and an abort will only increase network load.

8.7.2 Transport Protocol Data Units

The following PDU is used:

1. Abort PDU

8.7.3 Service primitives

The following service primitive is used:

1. TR-Abort

8.7.4 Procedure

There are three special cases of the abort procedure:

A) The sending WTP provider has not yet sent the message: the provider MUST discard the message from its memory.
B) The sending WTP provider has sent the message to the peer, or is in the process of sending the message: the provider MUST send the Abort PDU to the remote peer to discard all data associated with the transaction.
C) The receiving provider receives the Abort PDU: it generates the TR-Abort indication primitive and discards all transaction data.

When an Abort PDU is sent the reason for the abort is indicated in the abort reason field. There are two main types of aborts: User abort (USER) and Provider abort (PROVIDER). The user abort occurs when the WTP user has issued the TR-Abort request primitive. The provider abort occurs when there is an error in the WTP provider.

8.8 Transaction Identifier

8.8.1 Motivation

A transaction is uniquely identified by the socket pair (source address, source port, destination address and destination port) and the Transaction Identifier (TID). The Initiator increments the TID by one for every initiated transaction. This means that TIDs 1, 2 and 3 can go to server A, TIDs 4, 5 and 6 to server B and TIDs 7, 8 and 9 to server A.

The main use of the TID is to identify messages belonging to the same transaction. When a message is re-transmitted the TID is reused for the re-transmitted messages. A Responder MAY choose to remember the TID after an invoke message has been accepted and force TID verification in order to avoid replaying transactions. Also, the Initiator increments the TID by one for each transaction. This information can be used by the Responder to filter out new invoke messages from old and duplicated invoke messages: a new invoke always has a higher TID value.

Since transactions can be initiated simultaneously from both directions on the same socket association, the high order bit of the TID is used to indicate the direction of the transaction. The Initiator sets the high order bit to 0 in the Invoke PDU. Thereafter the high order bit is always inverted in the received TID before it is added to the response packet. By setting the high order bit of the TID field to 0 at the Initiator and 1 at the Responder, the Initiator can be guaranteed that the allocated TID will not collide with the remote entity.

The TID is 16-bits but the high order bit is used to indicate the direction. This means that the TID space is 2**15. The TID is an unsigned integer.

8.8.2 Procedure at the Responder

8.8.2.1 Variables

If the Responder caches old TID values for each different Initiator the old TID value is called LastTID. The TID in the received invoke message is called RcvTID.

8.8.2.2 Decisions when receiving a new invoke message

When the Responder receives an invoke message it takes one of the following actions depending on whether the Responder is caching old TID values or not, the characteristics of the underlying transport and the outcome of the TID test (described in the following chapter):

Table 8 Decisions when receiving new invoke message

Event	Condition	Action
TID test Fail	Underlying transport service can guarantee there are no duplicates (Note 1)	Start transaction
	Underlying transport service can NOT guarantee there are no duplicates	Invoke TID verification
TID test Ok		LastTID = RcvTID Start transaction
TIDnew flag set		LastTID = 0 Invoke TID verification
No cache	Responder caches the TID for each Initiator for the Maximum Packet Lifetime (MPL) of the network and it has not been re-booted during this time period and lost the information. If the invoke was not a new one, the Responder would have had the latest TID in itís cache.	Create new record for this Initiator LastTID = RcvTID Start transaction
	Responder does not cache TIDs (Note 2)	Invoke TID verification

Note 1) This is the case, for example, if a security layer is located under WTP and that can remove duplicates.
Note 2) This is not very efficient and SHOULD be avoided.

8.8.2.3 The TID test

One method of validating the TID is to use a window mechanism. The Responder MAY cache the last valid TID (LastTID) from each different Initiator. When the Responder receives a new invoke message it compares the TID in the invoke message (RcvTID) with the cached one. Let W be the size of the window. If W=2**14, it means that the boundary between two TID values occurs when they differ by 2**14, that is, half the TID space.

Table 9 TID test; RcvTID >= LastTID

RcvTID >= LastTID	
| RcvTID - LastTID |	TID test
0	Fail
<= W	Ok
> W	Fail (see 8.8.2.4)

Table 10 TID test; RcvTID < LastTID

RcvTID < LastTID	
| RcvTID - LastTID |	TID test
< W	Fail (see 8.8.2.4)
>= W	Ok

The above tables shows different results from the TID test. If the test succeeds it is guaranteed that the received invoke message is new and not an old delayed one. This is under the assumption that all messages have a Maximum Packet Lifetime (MPL), and that after MPL seconds it is guaranteed that there are no duplicate messages present in the network (see Note). Furthermore, it is assumed that the TID is not incremented faster than 2**14 steps in 2*MPL.

Note) For some networks types, the average Maximum Packet Lifetime MAY have a very high variance. For example, in a store-and-forward network like GSM SMS, a short message MAY reside in the SMS-C for a very long time, before it gets delivered to the destination. This fact MAY in some cases violate the correctness of the TID validation.

8.8.2.4 Reception of out-of-order invoke messages

Messages can arrive out-of-order. This means that even if the Initiator increments the TID by one for each transaction, a transaction with a lower TID value can arrive after a TID with a higher value. This MAY cause the TID test to fail and a TID verification to be started. This will not break the protocol, however, it will lead to degraded performance. One way to overcome this is to keep an array of TID values for past transactions. If the received TID is not in the array it can be accepted without any TID verification. This solution improves performance, but requires the Responder to maintain more information.

8.8.3 Procedure at the Initiator

8.8.3.1 Administration of TID

The Initiator is responsible for incrementing the TID by one for each transaction. This MUST NOT be done faster than 2**14 steps in 2*MPL.

8.8.3.2 Violating the monotonic property of the TID

There are cases when the Initiator MAY generate non-monotonic TID values, that is, the next TID MAY be smaller than the previous:

1. The Initiator has crashed and re-booted and randomly picked a smaller TID value than the previous.
2. The TID values have wrapped around the finite space. This can happen if, for example, the Initiator sends a transaction to Responder A, then sends 2**14 transactions to Responder B and finally returns to Responder A. The cached TID value at Responder A for this Initiator will now be smaller than current TID.

Neither of these two cases will break the protocol. However, TID verifications will be invoked and that will lead to lower efficiency.

In (1), if the Responder discards cached TID values after MPL seconds and the time to re-boot takes longer than that, the Responder will accept the new TID value without a TID verification (see 8.8.2.2). We have assumed that it will take longer time than 2*MPL to increment the TID 2**14 steps. However, if the Responder caches the TID value longer than for MPL seconds it will initiate a TID verification in this case.

The wraparound in (2) will be detected only if the Initiator caches the last sent TID to each Responder.

In both (1) and (2) excessive use of the TID verification mechanism SHOULD be avoided by setting the TIDnew flag in the Invoke PDU, i.e. when the initiator has received multiple subsequent verification requests from the responder the TIDnew flag should be set in the next transaction. This will invalidate the Responder's cached TID for the Initiator (see 8.8.2.2). When the Initiator uses the TIDnew flag it MUST NOT initiate any subsequent transaction until the TID verification has been completed. The reason for this is that the TIDnew MAY be delayed in the network. If, during that time period, transactions with higher TID are initiated, duplicates from these will get erroneously accepted when Responder has updated its cache with the lower TID in the TIDnew packet.

8.9 Transaction Identifier Verification

8.9.1 Motivation

The transaction identifier verification procedure is a three-way handshake. A three-way handshake between an Initiator (I) and a Responder (R) has the following steps:

(1) I → R This is the TID (Invoke PDU)
(2) I ← R Do you have an outstanding transaction with this TID? (Ack PDU)
(3) I → R Yes/No! (Ack PDU / Abort PDU)

The TID verification procedure is necessary to guarantee that the same invoke message is not accepted and delivered to the WTP user more the once, due to old duplicate packets.

The invoke message MUST NOT be delivered to the user before the TID verification procedure is completed successfully.

8.9.2 Protocol Data Units

The following PDUs are used:

1. Invoke PDU
2. Ack PDU
3. Abort PDU

8.9.3 Procedure

In the event that the Responder has received an Invoke PDU from an Initiator and has decided, using the rules for the Transaction Identifier procedure, to verify the TID, the following process is used.

The Responder sends an Ack PDU with Tve flag set indicating that it has received an invoke message with this TID.

When the Initiator receives the Ack PDU from the Responder it checks whether it has a corresponding outstanding transaction with this TID. In this case, the Initiator sends back an Ack PDU with TIDok flag set indicating that the TID is valid. This completes the three way handshake. If the Initiator does not have a corresponding outstanding transaction, it MUST abort the transaction by sending an Abort PDU with the Abort reason INVALIDTID.

Depending on the outcome of the TID verification WTP SHOULD take different actions. These are listed in the below table.

Table 11 Actions depending on result of TID verification

Result of TID verification	Condition	Action
Valid TID	TIDnew == True	Start transaction LastTID = RcvTID
	TIDnew == False	Start transaction LastTID = LastTID
Invalid TID		Abort transaction

The TIDnew flag is set in the invoke message and is used by the Initiator to invalidate the Responder's cache.

An example of this procedure can be found in chapter 11.5.

8.10 Transport Information Items (TPIs)

8.10.1 Motivation

The variable portion of the header in a WTP PDU MAY consist of Transport Information Items (TPIs). If not, the variable part of the header MUST be empty. The use of TPIs allows for future extensions of the protocol.

8.10.2 Procedure

All TPIs follow the general structure: TPI identity, TPI length and TPI data; the length can be zero. The following table lists the currently defined TPI and in which section they are explained:

Table 12 WTP Transport Information Items (TPIs)

Transport Information Item	Described in section
Error	"Transport Information Items (TPIs)" section 8.10
Info	"Information in Last Acknowledgement" section 8.4
Option	"Transmission of Parameters" section 0
Packet Sequence Number	"Segmentation and Re-assembly" section 8.13

A WTP provider without error ignore a TPI it does not implement, assuming the general TPI structure is used by all TPIs.

The error TPI can be used to inform the sender that an unsupported or erroneous TPI was received. When a WTP provider receives a TPI that is not supported, the WTP provider returns the Error TPI with the ErrorCode indicating "Unknown TPI" along with the identity of the unsupported TPI. When a WTP provider receives a supported TPI, but fails to understand the content of the TPI, the WTP provider returns the Error TPI with the ErrorCode indicating "Known TPI, unknown content", and the identity of the TPI and the first octet of the content included as argument.

Note that if an unsupported or erroneous TPI is received in the last message of a transaction, the receiver can not notify the sender of the event.

Transmission Of Parameters

8.10.3 Motivation

Protocol parameters can be transmitted between two WTP providers by using the Option TPI in the variable part of the PDU header.

No mandatory parameters have been defined.

Optional parameters used by the segmentation and re-assembly function are listed in 9.4.4.

8.10.4 Procedure

A WTP provider MAY support only a subset of all parameters. The parameters are transported in the variable part of the PDU header by using the Option TPI. The first octet of the Option TPI identifies the parameter and the following octets contains the value of the parameter. A WTP provider not supporting a parameter ignores it and returns the Error TPI.

8.11 Error Handling

8.11.1 Motivation

When an unrecoverable error is detected during the transaction, the transaction MUST be aborted. Currently no recovery mechanisms have been defined.

8.11.2 Protocol Data Units

The following PDU is used:

1. Abort PDU

8.11.3 Procedure

When an error occurs in the WTP provider during a transaction, the transaction MUST be aborted with an appropriate Abort reason and the local WTP user informed. The abort procedure is described in a separate section.

8.12 Version Handling

8.12.1 Motivation

A WTP provider receiving an invoke message with a higher version number than what is supported MUST abort the transaction.

8.12.2 Protocol Data Units

The following PDUs and parameters are used:

1. Invoke PDU
2. Abort PDU

8.12.3 Procedure

The Initiator indicates its version in the version field of the Invoke PDU.

If the Responder does not support the version it MUST return an Abort PDU with the Abort Reason set to WTPVERSIONONE. This indicates that the WTP provider supports version one of the WTP protocol.

8.13 Segmentation and Re-assembly (Optional)

8.13.1 Motivation

If the length of a message exceeds the MTU for the current bearer, the message can be segmented by WTP and sent in several packets. When a message is sent as a large number of small packets, the packets MAY be sent and acknowledged in groups. The sender can exercise flow control by changing the size of the packet groups depending on the characteristics of the network.

Selective re-transmission allows for a receiver to request one or multiple lost packets. The alternative is for the sender to re-transmit the entire message, which MAY include packets that have been successfully received. This function minimizes the number of packets sent by WTP.

This function is optional. If SAR is not implemented in WTP, this functionality has to be provided by another layer in the stack. For example, in IS-136 the SSAR layer handles SAR, in an IP network IP [RFC791] handles SAR and for GSM SMS/USSD SAR is achieved by using SMS concatenation [GSM0340]. The motivation for implementing WTP SAR is the selective re-transmission procedure, which MAY, if large messages are sent, improve the over-the-air efficiency of the protocol.

An example of this procedure can be found in chapter 11.6.

8.13.2 Procedure for Segmentation

For the sake of brevity only the procedure to segment an invoke message is described here (segmentation of a result message is identical except for the names of the PDUs.)

An invoke message which exceeds the MTU for the network is segmented into an ordered sequence of one Invoke PDU followed by one or more Segmented Invoke PDUs. The initial Invoke PDU has the implicit packet sequence number of zero, the following Segmented Invoke PDU has the packet sequence number one and all the following Segmented Invoke PDUs have packet sequence number that is one greater than the previous (n, n+1, n+2, etc). The Invoke PDU has an "implicit" packet sequence number since this number is not included as a field in the header. The client indicates in the Invoke PDU if the invoke message is segmented by clearing the TTR flag. If the invoke message is segmented, the server counts the Invoke PDU as packet number zero and waits for the following Segmented Invokes PDUs. The packet sequence number MUST NOT wrap. The packet sequence number field is 8 bits; and thus the maximum number of packets is 256.

8.13.3 Procedure for Packet Groups

The packets (Segmented Invoke PDUs and/or Segmented Result PDUs) are sent and acknowledged in groups. The sender MUST NOT send any new packets belonging to the same transaction until the previous packet group has been acknowledged. That is, packet groups are sent according to a stop-and-wait protocol. The sender determines the number of packets for each packet group. The size of a packet group SHOULD be decided with regards to the characteristics of the network and the device. No procedure for determining packet group size has been defined.

The packets in a packet group are sent in one batch. The last packet of the group has the GTR flag set. The last packet of the last packet group of the entire message has the TTR flag set. Since the first group is sent without knowing the status of

the receiver the number of packets SHOULD not be too large. When the receiver receives a packet that is not a GTR or TTR packet it MUST store the packet and wait for a new one.

When the receiver receives a packet with the GTR flag set it MUST check whether it has received all packets belonging to that packet group. If the complete packet group has been received the receiver returns an Ack PDU with the PSN TPI containing the Packet Sequence Number of the GTR packet. If one or more packets are missing the receiver returns a Nack PDU including the sequence number(s) of missing packet(s). The missing packets are re-transmitted with the original Packet Sequence Numbers but with the Re-transmission Indicator flag set. When the receiver has received the complete packet group, including those that were re-transmitted, it acknowledges the GTR packet.

When the receiver has received a complete packet group and the last packet has the TTR flag set, it SHOULD be able to re-assemble the complete message.

If the sender has not received an acknowledgment when the re-transmission timer expires, only the GTR/TTR packet is re-transmitted, not the entire packet group.

8.13.4 Procedure for Selective Re-transmission

When a GTR or TTR packet has been received and one or more packets of the packet group are missing, the WTP provider returns the Nack PDU with the sequence number of the missing packet(s). For example, if the receiver has received packet number 2, 3, 5 and 7, and packet number 7 has the GTR flag set, it returns a Nack PDU with packet numbers 4 and 6, indicating missing packets. The packet sequence number of the missing packets are contained in the header part of the Nack PDU.

If the Nack PDU is received with the number of missing packets field set to zero, this means that the entire packet group shall be re-transmitted.

The missing packets are re-transmitted with the original Packet Sequence Numbers. When the sender has re-transmitted the requested packets, it reverts to wait for the original acknowledgement (for the GTR or TTR packet).

When the receiver has received all packets it acknowledges the GTR or TTR packet according to the normal procedure, using the Ack PDU.

A WTP provider not supporting this function MUST re-transmit the entire message when one or multiple packets are requested for re-transmission.

When the GTR or TTR packet has been received and one or more packets of the group are missing, the WTP provider SHOULD wait for some period of time, such as ½ the median round-trip, before returning the Nack PDU with the sequence numbers of the missing packet(s). It the status of the group changes during the time, i.e. one of the missing packets is received, the waiting time SHOULD be reset.

9. Structure and Encoding of Protocol Data Units

9.1 General

A Protocol Data Unit, PDU, contains an integer number of octets and consists of:

a) the header, comprising:
    1. the fixed part
    2. the variable part
b) the data, if present

The fixed part of the headers contains frequently used parameters and the PDU code. The length and the structure of the fixed part are defined by the PDU code. The following PDU types are currently defined:

Table 13 WTP PDU Types

PDU Type	PDU Code
* NOT ALLOWED *	0x00 (Note 1)
Invoke	0x01
Result	0x02
Ack	0x03
Abort	0x04
Segmented Invoke	0x05 (Note 2)
Segmented Result	0x06 (Note 2)
Negative Ack	0x07 (Note 2)

The variable part is used to define less frequently used parameters. Variable parameters are carried in Transport Information Items, TPI.

The very first bit of the fixed header indicates whether the PDU has a variable header or not. The length of the fixed header is given by the PDU type. The variable header consists of TPIs. Every TPI has a length field for it's own length. The very first bit of each TPI indicates whether it is the last TPI or not.

Network Octet order for multi-octet integer values is “big-endian”. In other words, the most significant octet is transmitted on the network first followed subsequently by the less significant octets.

The left most bit (bit number 0) of an octet or a bit field is the most significant. Bit fields described first are placed in the most significant bits of the octet. The transmission order in the network is determined by the underlying transport mechanism

Note 1) If the first octet of a datagram is 0x00, it will be interpreted as if the datagram contains multiple concatenated PDUs. See section on Encoding of Concatenated PDUs.

Note 2) This PDU is only applicable if the optional Segmentation and Re-assembly function is implemented.

9.2 Common Header Fields

9.2.1 Continue Flag, CON

As the first bit of the fixed portion of the header, the Continue Flag indicates the presence of any TPIs in the variable part. If the flag is set, there are one or more TPIs in the variable portion of the header. If the flag is clear, the variable part of the header is empty.

This flag is also used as the first bit of a TPI, and indicates whether the TPI is the last of the variable header. If the flag is set, another TPI follows this TPI. If the flag is clear, the octet after this TPI is the first octet of the user data.

9.2.2 Group Trailer (GTR) and Transmission Trailer (TTR) flag

When segmentation and re-assembly is implemented the TTR flag is used to indicate the last packet of the segmented message, the GTR flag is used to indicate the last packet of a packet group.

Table 14 GTR/TTR flag combinations

GTR	TTR	Description
0	0	Not last packet
0	1	Last packet of message
1	0	Last packet of packet group
1	1	Segmentation and Re-assembly NOT supported

The default setting SHOULD be GTR=1 and TTR=1, that is, WTP segmentation and re-assembly not supported.

9.2.3 Packet Sequence Number

This is used by the PDUs belonging to the segmentation and re-assembly function. This number indicates the position of the packet in the segmented message.

9.2.4 PDU Type

The PDU Type field indicates what type of WTP PDU the PDU is (Invoke, Ack, etc). This provides information to the receiving WTP provider as to how the PDU data SHOULD be interpreted and what action is required.

9.2.5 Reserved, RES

All reserved bits are to be set to the value 0x00 unless otherwise specified.

9.2.6 Re-transmission Indicator, RID

Enables the receiver to differentiate between packets duplicated by the network and packets re-transmitted by the sender. In the original message the RID is clear. When the message gets re-transmitted the RID is set.

9.2.7 Transaction Identifier, TID

The TID is used to associate a packet with a particular transaction.

9.3 Fixed Header Structure

9.3.1 Invoke PDU

Table 15 Structure of Invoke PDU

Bit/Octet	0	1	2	3	4	5	6	7
1	CON	PDU Type = Invoke				GTR	TTR	RID
2 3	TID							
4	Version		TIDnew	U/P	RES	RES	TCL	

9.3.1.1 Transaction Class, TCL

The Initiator indicates the desired transaction class in the invoke message.

Table 16 Encoding of Class field

Class	TCL
0	0x00
1	0x01
2	0x10

The transaction classes are explained in separate chapter.

9.3.1.2 TIDnew flag

This is set when the Initiator has "wrapped" the TID value; that is, the next TID will be lower than the previous. When the Responder receives the Invoke PDU and the TIDnew flag is set, it invalidates its cached TID value for this Initiator.

9.3.1.3 Version

The current version in 0x00.

9.3.1.4 U/P flag

When this flag is set it indicates that the Initiator requires a User acknowledgement from the server WTP user. This means that the WTP user confirms every received message.

When this flag is clear the WTP provider MAY respond to a message without a confirmation from the WTP user.

9.3.2 Result PDU

Table 17 Structure of Result PDU

Bit/Octet	0	1	2	3	4	5	6	7
1	CON	PDU Type = Result				GTR	TTR	RID
2 3	TID							

9.3.3 Acknowledgement PDU

Table 18 Structure of Ack PDU

Bit/Octet	0	1	2	3	4	5	6	7
1	CON	PDU Type = Acknowledgement				Tve/Tok	RES	RID
2 3	TID							

9.3.3.1 Tve/Tok flag

In the direction from the reponder to the initiator the Tve (TID Verify) means: -"Do you have an outstanding transaction with this TID?". In the opposite direction the Tok (TID OK) flag means: -"I have an outstanding transaction with this TID!".

9.3.4 Abort PDU

Table 19 Structure of Abort PDU

Bit/Octet	0	1	2	3	4	5	6	7
1	CON	PDU Type = Abort				Abort type		
2 3	TID							
4	Abort reason							

9.3.4.1 Abort type and Abort reasons

Currently the following abort types are specified:

Table 20 WTP Abort Types

Abort type	Code	Description
Provider (PROVIDER)	0x00	The abort was generated by the WTP provider itself. The abort reason is specified below.
User (USER)	0x01	The abort was generated by the WTP user. The abort reason is provided to the WTP provider by the WTP user.

Abort reasons from the WTP provider

The following abort reasons are specified:

Table 21 WTP Provider Abort Codes

Abort reason (PROVIDER)	Code	Description
Unknown (UNKNOWN)	0x00	A generic error code indicating an unexpected error .

Abort reason (PROVIDER)	Code	Description
Protocol Error (PROTOERR)	0x01	The received PDU could not be interpreted. The structure MAY be wrong.
Invalid TID (INVALIDTID)	0x02	Only used by the Initiator as a negative result to the TID verification.
Not Implemented Class 2 (NOTIMPLEMENTEDCL2)	0x03	The transaction could not be completed since the Responder does not support Class 2 transactions.
Not Implemented SAR (NOTIMPLEMENTEDSAR)	0x04	The transaction could not be completed since the Responder does not support SAR.
Not Implemented User Acknowledgement (NOTIMPLEMENTEDUACK)	0x05	The transaction could not be completed since the Responder does not support User acknowledgements.
WTP Version zero (WTPVERSIONZERO)	0x06	Current version is 0. The initiator requested a different version that is not supported.
Capacity Temporarily Exceeded (CAPTEMPEXCEEDED)	0x07	Due to an overload situation the transaction can not be completed.
No Response (NORESPONSE)	0x08	A User acknowledgement was requested but the WTP user did not respond
Message too large (MESSAGETOOLARGE)	0x09	Due to a message size bigger than the capabilities of the receiver the transaction cannot be completed.

Abort reasons from the WTP user

The abort reasons from the WTP user are given to the local WTP provider in the T-TRAbort request primitive. The abort reason is specific to the WTP user. For example, if the WTP user is WSP, abort codes defined in [WSP] can be used.

9.3.5 Segmented Invoke PDU (Optional)

Bit/Octet	0	1	2	3	4	5	6	7
1	CON	PDU Type = Segmented Invoke				GTR	TTR	RID
2 **3**	TID							
4	Packet Sequence Number							

9.3.6 Segmented Result PDU (Optional)

Bit/Octet	0	1	2	3	4	5	6	7
1	CON	PDU Type = Segmented Result				GTR	TTR	RID
2 **3**	TID							
4	Packet Sequence Number							

9.3.7 Negative Acknowledgement PDU (Optional)

Bit/Octet	0	1	2	3	4	5	6	7
1	CON	PDU Type = Negative Ack				Reserved		RID
2 **3**	TID							

Bit/Octet	0	1	2	3	4	5	6	7
4	Number of Missing Packets = N							
5 … 4+N	Packet Sequence Number(s) of Missing Packets							

9.3.7.1 Number of Missing Packets

Indicates the requested number of missing packets. If 0x00, this means that the entire packet group shall be re-transmitted.

9.3.7.2 Packet Sequence Number(s) of Missing Packets

List of packet sequence number for the request packets.

9.4 Transport Information Items

9.4.1 General

The variable part of the PDU can consist of one or several Transport Information Items, TPIs. The length field of a TPI can be 2 or 8 bits.

The long TPI (8 bits length) has the following structure:

Table 22 Long TPI structure

Bit/Octet	0	1	2	3	4	5	6	7
1	CON	TPI Identity				1	RES	RES
2	TPI Length = N							
3 … 2+N	TPI Data							

The short TPI (2 bits length) is structured as

Table 23 Short TPI Structure

Bit/Octet	0	1	2	3	4	5	6	7
1	CON	TPI Identity				0	TPI Length = M	
2 … 1+M	TPI Data							

In the above tables, N=0..255 and M=0..3. The data field of the TPI MUST contain an integer number of octets. In theory the maximum length of a TPI is 255 octets, however, it is also limited by the MTU size of the bearer network and the number of, and length of, other TPIs in the same PDU header.

The following TPIs are currently defined:

Table 24 Encoding of TPIs

TPI	TPI Identity	Comment
Error	0x00	
Info	0x01	
Option	0x02	

TPI	TPI Identity	Comment
Packet Sequence Number (PSN)	0x03	Note 1

Note 1) This TPI is only applicable if the optional segmentation and re-assembly function is implemented.

9.4.2 Error TPI

The Error TPI is returned to the sender of an erroneous or unsupported TPI. Currently the following error codes have been defined:

Table 25 Encoding of Error TPI

Error	Code	Argument
Unknown TPI	0x01	TPI Identity of unknown TPI
Known TPI, unknown content	0x02	TPI Identity and first octet of content

Depending on the ErrorCode the Error TPI can have a different structure.

Table 26 Structure of Error TPI (UNKNOWN)

Bit/Octet	0	1	2	3	4	5	6	7
1	CON	TPI Identity = 0x00				0	TPI Length = 0x01	
2	ErrorCode = 0x01				Bad TPI Identity			

Table 27 Structure of Error TPI (KNOWN)

Bit/Octet	0	1	2	3	4	5	6	7
1	CON	TPI Identity = 0x00				0	TPI Length = 0x02	
	ErrorCode = 0x02				Bad TPI Identity			
3	First octet of TPI							

Note that this TPI is mandated to support by a WTP provider. Consequently, the WTP provider MUST also be able to recognise the general structure of a TPI.

9.4.3 Info TPI

This TPI is used to piggyback a small amount of data in the variable part of the PDU header. For example, the data can be performance measurements or statistical data.

The structure of the Info TPI is illustrated below.

Table 28 Structure of Info TPI

Bit/Octet	0	1	2	3	4	5	6	7
1	CON	TPI Identity				0	TPI Length = N	
2 … 1+N	Information							

The above table shows the Info TPI as short TPI. If more information MUST be sent, the long TPI can be used.

9.4.4 Option TPI

The Option TPI is used to transfer parameters between two WTP entities. The parameter carried in the Option TPI is valid for the lifetime of the transaction. The following options are currently defined:

Table 29 Encoding of Option TPI

Option	Identity	Description	Comment
Maximum Receive Unit	0x01	This parameter is used by the Initiator to advertise the maximum unit of data in bytes that can be received in the result	
Total Message Size	0x02	This parameter can be sent in the first packet of a segmented message to inform the receiver about the total message size in bytes	Note 1
Delay Transmission Timer	0x03	This parameter can be sent in the Ack PDU when a packet group is acknowledged. The receiver MUST NOT send the next packet group until the specified time has elapsed. The time is in 1/10 seconds.	Note 1
Maximum Group	0x04	This parameter can be used by either transaction party to advertise the maximum group size which can be received. The parameter indicates the maximum size in bytes of data in a single group.	
Current TID	0x05	This parameter may be sent with an Ack PDU when a 3-way-handshake is requested by the server, i.e. the Verify flag is set. The use of the parameter is optional, and the interpretation by the client implementation dependent. When used, the value shall be the value cashed by the server (LastTID).	
No Cached TID	0x06	This parameter may be sent with an Ack PDU when a 3-way-handshake is requested by the server, i.e. the Verify flag is set. The use of the parameter is optional, and the interpretation by the client implementation dependent. When used, the parameter indicated that there is no cached LastTID	

The structure of the Option TPI is illustrated below.

Table 30 Structure of Option TPI

Bit/Octet	0	1	2	3	4	5	6	7
1	CON	TPI Identity				0	TPI Length = N	
2	Option Identity							
3 … 1+N	Option Value							

Note 1) This parameter is only applicable if the optional segmentation and re-assembly function is implemented.

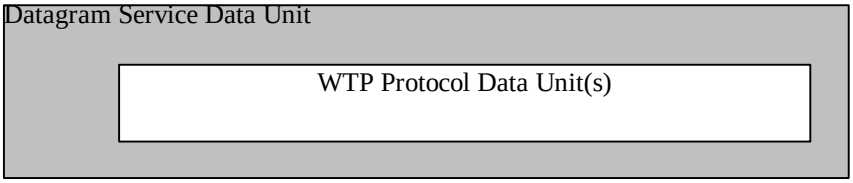
9.4.5 Packet Sequence Number TPI (Optional)

The Ack PDU does not have a Packet Sequence Number (PSN) field. When Segmentation and Re-assembly is used this TPI is attached to the variable part of the Ack PDU header. The PSN included in the Ack PDU is the PSN of the acknowledged packet (GTR or TTR packet).

Bit/Octet	0	1	2	3	4	5	6	7
1	CON	TPI Identity = PSN TPI				0	Length = 0x01	
2	Packet Sequence Number							

9.5 Structure of Concatenated PDUs

One or more WTP Protocol Data Units (PDUs) MAY be contained in one datagram Service Data Unit (SDU). This is illustrated below.



The following table represents a datagram SDU with one WTP PDU. The PDU including header and data is N octets.

Table 31 WTP PDU Without Concatenation

Bit/Octet	0	1	2	3	4	5	6	7
1 … N	WTP PDU							

The following table represents two WTP PDUs concatenated in the same SDU of the bearer network. The first PDU is N octets and the second is M octets.

Table 32 Concatenated WTP PDUs

Bit/Octet	0	1	2	3	4	5	6	7
1	Concatenation Indicator = 0x00							
2	0	WTP PDU Length = N						
3 … N+2	WTP PDU							
N+3	0	WTP PDU Length = M						
N+4 … N+M+3	WTP PDU							

The concatenation indicator is used to indicate that the SDU contains multiple WTP PDUs. The number of PDUs is limited only by the maximum size of the SDU.

The PDU Length field can be 7 or 15 bits. If the first bit in the PDU Length field is set, the length field is 15 bits, if not, it is 7 bits. This means that the PDU Length field takes up 8 or 16 bits depending on whether the first bit is set or not. In the above table the first bit is 0 and thus the length field is 7 bits.

10. State Tables

10.1 General

This chapter defines state tables for the core WTP protocol without the optional segmentation and re-assembly function.

10.2 Event Processing

The WTP provider initiating a transaction is called the Initiator. The WTP provider responding to an initiated transaction is called the Responder. An implementation of the WTP protocol is not required to have both Initiator and Responder functionality. For example, if the WTP user is the WSP client protocol the WTP provider MAY only support initiation of transactions, that is, no Responder functionality. See WTP conformance clause for details on what MUST be implemented in order to conform to the standard.

The interface to the next higher layer is defined by the WTP service primitives. The next lower layer is typically a datagram service and the only service primitives are the UnitData indication and requests. The request and response service primitives from the next higher layer together with indication primitive from the next lower layer are termed *events*. If multiple PDUs are concatenated in the SDU from the next lower layer, they MUST be separated and dispatched to the transactions. In addition to the external events, there will also be internal events such as timer expirations and errors.

An event is validated before it is processed. The following tests are performed, and if no action is taken, the event is processed according to the state tables.

Table 33 Test of incoming events

Test	Action
UnitData.ind on the Responder: Invoke PDU	Create a new transaction
UnitData.ind on the Initiator: Ack PDU with the TIDve flag set, no matching outstanding transaction	Send Abort PDU (INVALIDTID)
UnitData.ind: Ack PDU, Result PDU or Abort PDU, no matching outstanding transaction	Ignore
Illegal PDU type or erroneous header structure	Refer to entry for `RcvErrorPdu' in state tables
Buffer overflow or out-of-memory errors	Send Abort PDU (CAPTEMPEXCEED)
UnitData.ind on the Responder: Invoke PDU requesting Class 2 transaction and Class 2 is not supported	Send Abort PDU (NOTIMPLEMENTEDCL2)
UnitData.ind on the Responder: Invoke PDU using SAR and SAR is not supported	Send Abort PDU (NOTIMPLEMENTEDSAR)
UnitData.ind on the Responder: Invoke PDU requesting User acknowledgement and User acknowledgement is not supported	Send Abort PDU (NOTIMPLEMENTEDUACK)
UnitData.ind on the Responder: Invoke PDU with Version != 0x00	Send Abort PDU (WTPVERSIONONE)
UnitData.ind on the Responder: Invoke PDU when no more transaction requests can be accepted	Ignore.

10.3 Actions

10.3.1 Timers

The following timer actions can be used in the state tables:

Start timer, <value>
Starts the timer with the specified interval value. If the timer is already running, it is re-started with the new value.

Stop timer
Stop the timer without generating an event.

10.3.2 Counters

The following counter actions can be used in the state tables:

Reset counter
Set the counter to zero.

Increment counter
Increment the counter with one.

10.3.3 Messages

The following message actions can be used in the state tables:

Queue (Time T)
Queuing a PDU causes it to be queued for eventual delivery. The message MUST NOT be queued for longer time than T time units and is queued only until the already started timer T expires. The timer is not restarted

Send
Sending a PDU causes it and any queued PDUs to be sent immediately.

The queuing mechanism is used to concatenate messages from different transactions. This can be seen as a concatenation layer that operates below the transaction state machine. The realization of the concatenation layer is implementation dependent and not specified.

10.4 Timers, Counters and Variables

10.4.1 Timers

The following timers are used by WTP:

Table 34 WTP Timers

Timer	Description
Transaction timer	Each transaction has a timer associated with it. The timer is used for both the retry interval, acknowledgement interval and wait timeout interval.

A timer can be started with different timer values depending on the type of transaction and the current state of the transaction. Timer values are grouped according to their purpose. This is shown in the following table.

Table 35 WTP Timer Intervals

Timer interval (name)	Description
Acknowledgement interval (A)	This sets a bound for the amount of time to wait before sending an acknowledgement.
Retry interval (R)	This sets a bound for the amount of time to wait before re-transmitting a PDU.
Wait timeout interval (W)	This sets a bound for the amount of time to wait before state information about a transaction is released. Only Class 2 Initiator and Class 1 Responder

The Retry interval MAY be implemented as an array with the re-transmission counter as an index, R[RCR]. An exponential back off algorithm can be implemented by populating R[] with exponentially increasing values.

The value of a timer interval depends on the following parameters:

- The characteristics of the bearer network
- The transaction class
- The state of the transaction (which message is being re-tried or acknowledged)

Timer interval values for different bearer networks can be found in Appendix A.

10.4.2 Counters

The following counters are used by the WTP:

Table 36 WTP Counters

Counter (name)	Description
Re-transmission Counter (RCR)	This set a bound for the maximum number of re-transmissions of any PDU. The max value is defined as RCR_MAX.
Acknowledgment Expiration Counter (AEC)	This sets a bound for the maximum number of times the transaction timer, initialized with the acknowledgement interval, is allowed to expire and be re-started before the transaction is aborted. The max value is defined as AEC_MAX.

10.4.3 Variables

The following variables are used by WTP at the Initiator and Responder.

Table 37 WTP Variables

WTP variables			
Variables	**Type**	**Description**	
GenTID	Uint16	The TID to use for the next transaction. Incremented by one for every initiated transaction.	Global Only Initiator
SendTID	Uint16	The TID value to send in all PDUs in this transaction	One per transaction
RcvTID	Uint16	The TID values expected to receive in every PDU in this transaction. RcvTID = SendTID XOR 0x8000	One per transaction

LastTID	Uint16	The last received TID from a certain remote host	One per remote host Only Responder
HoldOn	BOOL	True if HoldOn acknowledgement has been received	One per class 2 transaction
Uack	BOOL	True if User Acknowledgement has been requested for this transaction	One per transaction

The Uint16 type is an unsigned 16-bit integer. The BOOL type is an boolean value that only can take the value of True or False.

10.5 WTP Initiator

WTP Initiator NULL			
Event	**Condition**	**Action**	**Next State**
TR-Invoke.req	Class == 2 | 1	SendTID = GenTID Send Invoke PDU Reset RCR Start timer, R [RCR] Uack = False	RESULT WAIT
	Class == 2 | 1 UserAck	SendTID = GenTID Send Invoke PDU Reset RCR Start timer, R [RCR] Uack = True	
	Class == 0	SendTID = GenTID Send Invoke PDU	NULL

WTP Inititator RESULT WAIT			
Event	**Condition**	**Action**	**Next State**
TR-Abort.req		Abort transaction Send Abort PDU (USER)	NULL
RcvAck	Class == 2	Stop timer Generate T-TRInvoke.cnf HoldOn = True	RESULT WAIT
	Class == 1	Stop timer Generate T-TRInvoke.cnf	NULL
	TIDve Class == 2 | 1	Send Ack(TIDok) Increment RCR Start timer, R [RCR]	RESULT WAIT
RcvAbort		Abort transaction Generate TR-Abort.ind	NULL
RcvErrorPDU		Abort Transaction Send Abort PDU (PROTOERR) Generate TR-Abort.ind	NULL
TimerTO_R	RCR < MAX_RCR	Increment RCR Start timer R [RCR] Send Invoke PDU	RESULTWAIT
	RCR < MAX_RCR	Increment RCR Start timer R [RCR] Send Invoke PDU	RESULT WAIT
	RCR < MAX_RCR, Ack(TIDok) already sent	Increment RCR Start timer R [RCR] Send Ack(TIDok)	RESULT WAIT
	RCR == MAX_RCR	Abort transaction Generate TR-Abort.ind	NULL
RcvResult	Class == 2 HoldOn == True	Stop timer Generate TR-Result.ind Start timer, A	RESULT RESP WAIT
	Class == 2 HoldOn == False	Stop timer Generate TR-Invoke.cnf Generate TR-Result.ind Start timer, A	

WTP Inititator RESULT RESP WAIT (class 2 only)			
Event	**Condition**	**Action**	**Next State**
TR-Result.res		Queue(A) Ack PDU Start timer, W	WAIT TIMEOUT
	ExitInfo	Queue(A) Ack PDU with Info TPI Start timer, W	
RcvAbort		Abort transaction Generate T-TRAbort.ind	NULL

WTP Inititator RESULT RESP WAIT (class 2 only)			
Event	**Condition**	**Action**	**Next State**
TR-Abort.req		Abort transaction Send Abort PDU (USER)	
RcvErrorPDU		Abort Transaction Send Abort PDU (PROTOERR) Generate TR-Abort.ind	NULL
RcvResult		Ignore	RESULT RESP WAIT
TimerTO_A	AEC < AEC_MAX	Increment AEC Start timer, A	RESULT RESP WAIT
	AEC == AEC_MAX	Abort transaction Send Abort PDU (NORESPONSE)	NULL
	Uack == False	Queue(A) Ack PDU Start timer, W	WAIT TIMEOUT

WTP Inititator WAIT TIMEOUT (class 2 only)			
Event	**Condition**	**Action**	**Next State**
RcvResult	RID=0	Ignore	WAIT TIMEOUT
RcvResult	RID=1	Send Ack PDU	WAIT TIMEOUT
RcvResult	RID=1, ExitInfo	Send Ack PDU with info TPI	WAI T TIMEOUT
RcvAbort		Abort transaction Generate T-TRAbort.ind	NULL
RcvErrorPDU		Abort Transaction Send Abort PDU (PROTOERR) Generate TR-Abort.ind	NULL
TimerTO_W		Clear Transaction	NULL
TR-Abort.req		Abort transaction Send Abort PDU (USER)	NULL

10.6 WTP Responder

WTP Responder LISTEN			
Event	**Condition**	**Action**	**Next State**
RcvInvoke	Class == 2 | 1 Valid TID U/P flag	Generate TR-Invoke.ind Start timer, A Uack = True	INVOKE RESP WAIT
	Class == 2 | 1 Valid TID	Generate TR-Invoke.ind Start timer, A Uack = False	
	Class == 0	Generate TR-Invoke.ind	LISTEN
	Class == 2 | 1 Invalid TID	Send Ack(TIDve)	TIDOK WAIT
RcvErrorPDU		Send Abort PDU (PROTOERR)	LISTEN

WTP Responder TIDOK WAIT			
Event	**Condition**	**Action**	**Next State**
RcvAck	Class == 2 | 1 TIDok	Generate TR-Invoke.ind Start timer, A	INVOKE RESP WAIT
RcvErrorPDU		Send Abort PDU (PROTOERR) Abort Transaction	LISTEN
RcvAbort		Abort transaction	LISTEN
RcvInvoke	RID=0	Ignore	TIDOK WAIT
	RID=1	Send Ack(TIDve)	TIDOK WAIT

WTP Responder INVOKE RESP WAIT			
Event	**Condition**	**Action**	**Next State**
TR-Invoke.res	Class == 1 ExitInfo	Queue(A) Ack PDU with InfoTPI Start timer,W	WAIT TIMEOUT
	Class == 1	Queue(A) Ack PDU Start timer, W	
	Class == 2	Start timer, A	RESULT WAIT
TR-Result.req		Reset RCR Start timer R[RCR] Send Result PDU	RESULT RESP WAIT
TR-Abort.req		Abort transaction Send Abort PDU (USER)	LISTEN
RcvAbort		Generate TR-Abort.ind Abort transaction	LISTEN
RcvInvoke		Ignore	INVOKE RESP WAIT
RcvErrorPDU		Abort Transaction Send Abort PDU (PROTOERR) Generate TR-Abort.ind	LISTEN
TimerTO_A	AEC < AEC_MAX	Increment AEC Start timer, A	INVOKE RESP WAIT
	AEC == AEC_MAX	Abort transaction Send Abort PDU (NORESPONSE)	LISTEN
	Class == 1 Uack == False	Queue(A) Ack PDU Start timer, W	WAIT TIMEOUT
	Class == 2 Uack == False	Send Ack PDU	RESULT WAIT

WTP Responder RESULT WAIT (class 2 only)			
Event	**Condition**	**Action**	**Next State**
TR-Result.req		Reset RCR Start timer, R[RCR] Send Result PDU	RESULT RESP WAIT
RcvInvoke	RID=0	Ignore	RESULT WAIT
	RID=1	Ignore	RESULT WAIT
	RID=1, Ack PDU already sent	Resent Ack PDU	RESULT WAIT
RcvErrorPDU		Abort Transaction Send Abort PDU (PROTOERR) Generate TR-Abort.ind	LISTEN
TR-Abort.req		Abort transaction Send Abort PDU (USER)	LISTEN
RcvAbort		Generate T-TRAbort.ind Abort transaction	LISTEN
TimerTO_A		Send Ack PDU	RESULT WAIT

WTP Responder RESULT RESP WAIT (class 2 only)			
Event	**Condition**	**Action**	**Next State**
TR-Abort.req		Abort transaction Send Abort PDU (USER)	LISTEN
RcvAbort		Generate T-TRAbort.ind Abort transaction	LISTEN
RcvAck		Generate TR-Result.cnf	LISTEN
RcvErrorPDU		Abort Transaction Send Abort PDU (PROTOERR) Generate TR-Abort.ind	LISTEN
TimerTO_R	RCR < MAX_RCR	Increment RCR Send Result PDU Start timer, R [RCR]	RESULT RESP WAIT
	RCR == MAX_RCR	Generate T-TRAbort.ind Abort transaction	LISTEN

WTP Responder WAIT TIMEOUT (class 1 only)			
Event	**Condition**	**Action**	**Next State**
RcvInvoke	RID=0	Ignore	WAIT TIMEOUT
RcvInvoke	RID=1	Send Ack PDU	WAIT TIMEOUT
RcvInvoke	RID=1, ExitInfo	Send Ack PDU with Info TPI	WAIT TIMEOUT
RcvErrorPDU		Abort Transaction Send Abort PDU (PROTOERR) Generate TR-Abort.ind	LISTEN
RcvAbort		Abort transaction Generate T-TRAbort.ind	LISTEN
TimerTO_W		Clear Transaction	LISTEN
TR-Abort.req		Abort transaction Send Abort PDU (USER)	LISTEN

11. Examples of Protocol Operation

11.1 Introduction

The examples in this chapters attempts to illustrate and clarify how the protocol operates. For the sake of brevity, only header fields relevant for the specific example are included in the diagrams. Each flag in the Flag field of the PDU header is indicated by one character. The below table shows the different characters that can appear in the examples.

Table 38 Abbreviations Used in the Examples

Abbreviation	Meaning
N	TIDnew flag is set
V	TIDve flag is set
O	TIDok flag is set
U	U/P flag is set
G	GTR flag is set
T	TTR flag is set
TG	Both TTR and GTR flags are set to indicate that SAR is not supported
RID = X	Re-transmission Indicator is X
TID = N	Transaction Identifier is N
c0	The TCL field indicates class 0 transaction
c1	The TCL field indicates class 1 transaction
c2	The TCL field indicates class 2 transaction

Parameters like Abort reason and Error codes are written in clear text, and so are TPIs. For Transaction Identifiers N* is N with the high order bit set; if N = 0x0000 then N* = 0x8000.

11.2 Class 0 Transaction

11.2.1 Basic Transaction

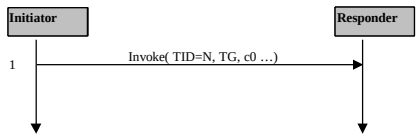


Figure 3 Basic Class 0 Transaction

1. The Initiator initiates a class 0 transaction (c0).

11.3 Class 1 Transaction

11.3.1 Basic Transaction

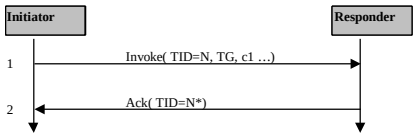


Figure 4 Basic Class 1 Transaction

1. The Initiator initiates a class 1 transaction (c1).
2. The Responder acknowledges the received invoke message.

11.4 Class 2 Transaction

11.4.1 Basic Transaction

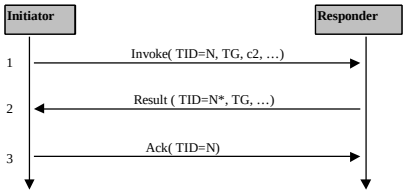


Figure 5 Basic Class 2 Transaction

1. The Initiator initiates a class 2 transaction (c2).
2. The Responder waits for the invoke message to be processed and implicitly acknowledges the invoke message with the Result.
3. The Initiator acknowledges the received result message.

11.4.2 Transaction with “hold on” Acknowledgement

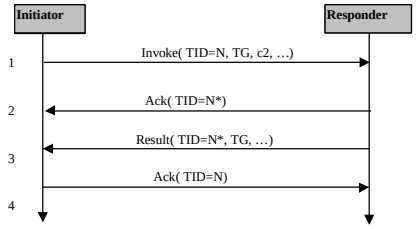


Figure 6 Class 2 Transaction with "hold on" acknowledgement.

1. The Initiator initiates a class 2 transaction (c2).
2. The Responder waits for the invoke message to be processed. The acknowledgement timer at the Responder expires and an "hold-on" acknowledgement is sent to prevent the Initiator from re-transmitting the invoke message.
3. The result is sent to the Initiator
4. The Initiator acknowledges the received result message.

11.5 Transaction Identifier Verification

11.5.1 Verification Succeeds

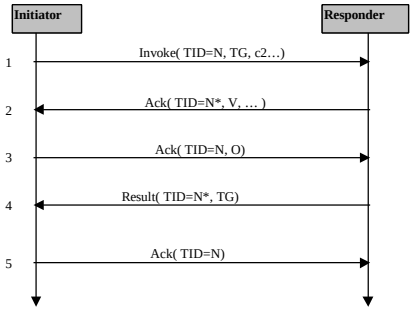


Figure 7 Verification Succeeds

The Responder receives a new invoke message and the TID test fails, this causes the Verification procedure to be invoked. The Responder returns an acknowledgement to the Initiator for a verification of whether it has an outstanding transaction with this TID. In this example, the Initiator has an outstanding transaction with the TID and acknowledges the verification.

11.5.2 Verification Fails

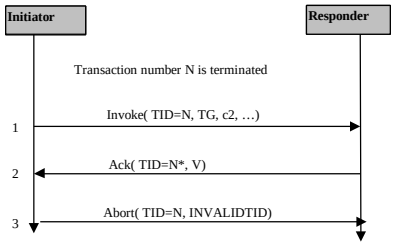


Figure 8 Verification Fails

The invoke message with TID=N is duplicated in the network, or has been delayed. When it arrives, transaction N has already been terminated and the Responder asks the Initiator to verify the transaction. The Initiator aborts the transaction by sending an Abort.

11.5.3 Transaction with Out-Of-Order Invoke

An invoke message is delayed in the network. When the message finally arrives to the Responder, the Responder has cached a higher TID value. The Responder initiates a Verification in order to check whether the Initiator still has an invoke message with TID=N outstanding.

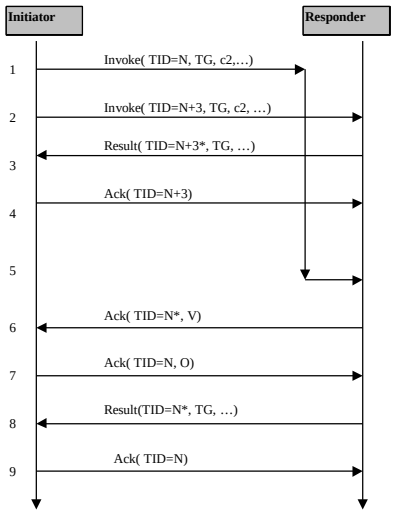


Figure 9 Delayed Invoke Message

Note that the Responder must not replace it’s cached TID value (N+3) with the lower TID value (N). If the cached TID is moved backwards, old duplicates with higher TID values will erroneously get accepted.

11.6 Segmentation and Re-assembly

This example illustrates a Class 2 transaction using segmentation. The Invoke is segmented and sent in five packets in two packet groups.

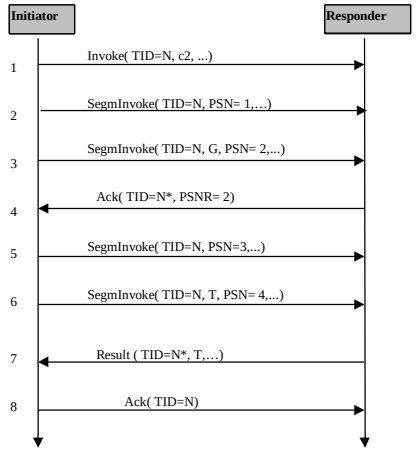


Figure 10 Segmentation of invoke message

The Initiator starts off by sending the first three packets in one batch. The last packet has the GTR flag to trigger an acknowledgement from the Responder. Once the acknowledgement is received by the Initiator the last two packets of the message are sent. The final message has the TTR flag set. After some time, the Responder sends back the result to the Initiator. The Initiator acknowledges the result and the transaction is finished.

Note that the PSN TPI is used for the Packet Sequence Number in the Ack PDU.

11.6.1 Selective Re-transmission

This example illustrates a Class 1 transaction using segmentation. One of the packets in the first packet group is lost and the Responder has to request the packet to be re-transmitted.

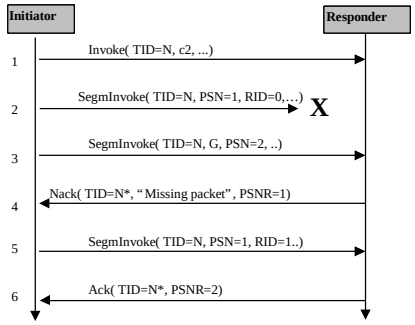


Figure 11 Selective re-transmission

The Initiator starts off by sending the first three packets. The second packet is lost. When the Responder receives the packet with the GTR flag set, it attempts to re-assemble the packet group but fails due to the one missing packet. The Responder returns a Nack to request the missing packet. The Initiator re-transmit the missing packet. The re-transmitted packet has the RID flag set. Once the missing packet has been received by the Responder the message is acknowledged and the transaction is finished.

Note that the PSN TPI is used for the Packet Sequence Number in the Ack PDU.

Appendix A. Default Timer and Counter Values

The timers are initial estimates and have not yet been verified.

The timer values in the tables below are expressed in seconds. The counters are expressed in times an event happens.

GSM SMS

The maximum round-trip time is assumed to be 40 seconds (while a median round-trip time is about 10 seconds), and the timer values are thus suggested to be:

Timer interval	Type	Without User Ack.	With User Ack.
Base Acknow. interval (A)	B_A	10	10
- Short	S_A	0	5
- Long	L_A	20	20
Base Retry interval (R)	B_R	60	60
- Short	S_R	35	40
- Long	L_R	70	70
- Group	G_R	45	45
Wait timeout interval (W)	W	300	300

Counter name	Value for stack acks	Value for user acks
Max Retransmissions	4	4
Max Ack timer Expiration	4	4

GSM USSD

The maximum round-trip time is assumed to be 5 seconds, and the timer values are thus suggested to be:

Timer interval	Type	Without User Ack.	With User Ack.
Base Acknow. Interval (A)	B_A	10	10
- Short	S_A	0	5
- Long	L_A	10	10
Base Retry Interval;	B_R	20	20
- Short	S_R	14	14
- Long	L_R	20	20
- Group	G_R	10	10
	W	60	60

Counter name	Value for stack acks	Value for user acks
Max Retransmissions	4	4
Max Ack timer Expiration	4	4

Bearers supporting IP (Circuit switched, CDPD...)

The maximum round-trip time is assumed to be 3 seconds, and the timer values are thus suggested to be:

Timer interval	Type	Without User Ack.	With User Ack.
Base Acknow. interval (A)	B_A	2	2
- Short	S_A	0	1
- Long	L_A	4	4
Base Retry interval (R)	B_R	5	5
- Short	S_R	3	4
- Long	L_R	7	7
- Group	G_R	3	3
Wait timeout interval (W)	W	40	40

Counter name	Value for stack acks	Value for user acks
Max Retransmissions	8	8
Max Ack timer Expiration	6	6

Timer Usage

There are a number of timer interval with similar behavior, but different values. These timers are defined to enable an optimal use of the available bandwidth. Chapter 10 refers to the abstract timer intervals A and R. These are mapped to the real interval values as defined in this section.

Message type	Class 2	Class 1
Invoke message	B_R	S_R
Hold on acknowledgement	B_A	-
Result message	L_R	-
Last acknowledgement	L_A	S_A
Last packet of packet group	G_R	G_R

For Class 0 no timer values are applicable.

Appendix B. Implementation Note

The following implementation notes are provided to identify areas where implementation choices may impact the performance and effectiveness of the WTP protocol. These notes provide guidance to implementers of the protocols.

C.1 Extended Timers for Large Messages

The Wireless Transaction Protocol is using Retransmission timers both to ensure reliable delivery of data to the receiver as well as to create a predictable behaviour in a lossy environment. The default timers are defined for relatively small transmissions. The protocol can manage also large data amounts, but then the timer values need to be adjusted. This implementation note suggests a scheme for recalculation of the timer intervals.

When large messages are transmitted from sender to receiver the transmission time for the complete message (or group) can become larger than the default value for the retransmission timer. The value thus has to be recalculated in order to avoid unnecessary retransmissions.

The new timer value to be used with both large datagrams as well as segmented transactions can be (implementation dependent) calculated according to the algorithm below:

rt = T + n * M

where

rt – recalculated retransmission timer value
T – Original timer value
n – estimated number of fragments (estimate made by protocol layer implementation)
M – bearer dependent value of ½ median roundtrip (estimate defined below)

The same algorithm is used both for large datagrams that will be fragmented at a lower protocol layer, such as in the WDP protocol, as well as for segmented messages where the WTP layer divides a group/message into multiple segments. The factor n is estimated in the former case and calculated in the latter case.

The following values for M can be used

Bearer	Median ½ roundtrip in seconds
SMS	5
USSD	3
IP	0.2

The value for IP represents a unit of 1 kbyte. However, the timer value can be rounded to the nearest smaller integer.

Appendix C. History and Contact Information

Document history		
Date	**Status**	**Comment**
11-June-199	Draft	Integration of the approved corrigendum (WTP-Corrigendum-7-May-1999)
7-May-1999	Specification	Second Version
29-April-1998	Specification	First version.
Contact Information http://www.wapforum.org. technical-comments@wapforum.org		